ENTITY-CENTRIC SEARCH: QUERYING BY ENTITIES AND FOR ENTITIES

BY

MIANWEI ZHOU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Doctoral Committee:

Professor Kevin C. Chang, Chair
Professor Jiawei Han
Professor ChengXiang Zhai
Doctor Kuansan Wang, Microsoft Research

Abstract

The immense scale of the Web has rendered itself as a huge repository storing information about various types of entities (*e.g.*, persons, locations, products, companies). Much of information retrieval operations on the Web nowadays are about entities, *i.e.*, entity-centric, for example, finding cameras that have black color and high resolution, keeping track of important events of a favorite celebrity, etc.. However, without modeling the concept of entities, most search engines still take the page view of the Web data, which accepts keyword queries as input, and aims at finding documents that are relevant to the queries. Therefore, in my thesis, I propose to study entity-centric search towards facilitating various types of entity-related information search operations. When the concept of entity is involved in a search operation, people are usually interested in finding more information about some known entities, or exploring unknown entities that satisfy certain information needs. I propose to categorize such operations into two categories: querying by entities and querying for entities. As the objective of my thesis, I aim at building a general framework to facilitate these different types of entity-centric search operations. First, in query by entities, I propose to address the entity-centric document filtering problem, which, towards better characterizing the target entity, adopts its entity identification page (*e.g.*, Wikipedia page) as input to identify its relevant documents. Second, in query for entities, witnessing many different ad-hoc efforts for exploiting fine granularity entity information across Web text, *e.g.*, typed-entity search, question answering, I propose to build a general data-oriented content query system, which distills their essential capabilities and supports "content querying" for finding various entity data in the text. Third, I study the relational entity search problem, *i.e.*, given a query entity, how to search entities that match a desired relation – since the search operation involves entities in both input query and output result, it belongs to querying by entities and for entities at the same time. The results we obtained so far show clear promise of entity-centric search in its usefulness, effectiveness and efficiency.

Table of Contents

Chapter 1 Introduction **1**

Chapter 2 Query By Entities: Entity-Centric Document Filtering **5**
2.1 Introduction 5
2.2 Related Work 8
2.3 Entity-Centric Filtering 10
2.3.1 Problem & Framework 10
2.3.2 Challenge: Learning across Entities 12
2.3.3 Key Insight: Features of Features 13
2.4 Meta-Feature Based Feature Mapping 14
2.4.1 Continuous Linear Mapping Model 15
2.4.2 Discrete Boosting Mapping Model 15
2.4.3 LinearMapping versus BoostMapping 19
2.5 Experiment 20
2.5.1 Experiment Setting 21
2.5.2 Quantitative Evaluation 22
2.5.3 Case Study 27

Chapter 3 Query By Entities: Cross-Task Document Scoring **28**
3.1 Introduction 28
3.2 Related Work 32
3.3 Cross-Task Document Scoring 33
3.3.1 Problem: Cross-Task Document Scoring 34
3.3.2 Challenge: Learning to Adapt Keyword Contribution 34
3.3.3 Insight: Keyword Scoring Principle 35
3.3.4 Abstract: Feature Decoupling 36
3.4 Tree-Structured Restricted Boltzmann Machine 38
3.4.1 Requirement: Inferred Sparsity & Distant Supervision 39
3.4.2 Proposal: Two-Stage Scoring Model 40
3.4.3 Solution: Tree-Structured Restricted Boltzmann Machine 41
3.5 Experiment 46
3.5.1 Experiment Setting 46
3.5.2 Quantitative Evaluation 47
3.5.3 Case Study 51
3.6 Framework Generalization 52

Chapter 4 Query For Entities: Data-Oriented Content Query System **55**
4.1 Introduction 55
4.2 Related Work 57
4.3 Data Model 58
4.4 Content Query Language (CQL) 59
4.4.1 Design Principle 60

4.4.2 CQL Specification . . . 61
4.4.3 Pattern & Weighting . . . 63
4.4.4 Scoring Specification . . . 65
4.4.5 Data Type Definition . . . 66
4.5 Indexing & Query Processing . . . 67
4.5.1 Indexing Design . . . 67
4.5.2 Index Configuration . . . 70
4.5.3 Query Processing . . . 71
4.6 Experimental Results . . . 74
4.6.1 Case Study . . . 74
4.6.2 Time Cost Analysis . . . 77
4.6.3 Space Cost Analysis . . . 78

Chapter 5 Query By and For Entities: Relational Entity Search . . . 79
5.1 Introduction . . . 79
5.2 Related Work . . . 82
5.3 Exploring Noisy Redundancy for Relational Entity Search . . . 83
5.3.1 Relational Entity Search Framework . . . 84
5.3.2 Distantly Supervised Ranking . . . 87
5.4 Pattern-Based Filter Network . . . 88
5.4.1 Noise Filtering . . . 90
5.4.2 Evidence Aggregation . . . 90
5.4.3 Objective of PFNet . . . 91
5.4.4 Model Learning . . . 93
5.5 Experiment . . . 95
5.5.1 Experiment Setting . . . 95
5.5.2 Performance Comparison with Baselines . . . 96
5.5.3 Influence of Redundancy on Ranking Performance . . . 99
5.5.4 Performance Comparison with Different Feature Size . . . 102
5.5.5 Case Study . . . 103

Chapter 6 Conclusion . . . 105
6.1 Summary: Contributions of My Thesis . . . 105
6.2 Goal: A Practical Entity-Centric Search System . . . 106
6.2.1 Future Work 1: Query Intent Analysis . . . 108
6.2.2 Future Work 2: Informative Entity Result . . . 109
6.2.3 Future Work 3: Other Types of Entity-Centric Search . . . 109

References . . . 111

Chapter 1

Introduction

The immense scale of the Web has rendered itself as a huge repository storing information about various types of entities, for example, daily events for persons, specification for products, geographical information for locations, *etc.*. With the explosive growth of entity information on the Web, we also observe rapidly increasing entity-centric information needs from end users. As reported by Kumar *et al.* [40], 52.9% of Web search queries are entity-oriented; Guo *et al.* [28] also show that 71% of Web search queries contain named entities.

Taking the conventional page view of the Web data, traditional IR frameworks take a few number of keywords as input, and return a list of relevant documents as the result. Without being aware of the concept of entity in users' information needs, such frameworks have limitations in handling entity-centric search operations. For example, given query "treatment of anxiety disorder," without identifying that "anxiety disorder" is the entity name, and "treatment" is the target relation, the search engine fails to know that the result is more likely to appear in a descriptive page of "anxiety disorder," and and the keyword of "treatment" could be replaced by "therapy," "heal," *etc.* Furthermore, in many entity-centric queries, users are interested in retrieving entities as the result; however, traditional search engines only support returning a list of documents, which is inconvenient for users as it requires users to check the document content to get the entity answer.

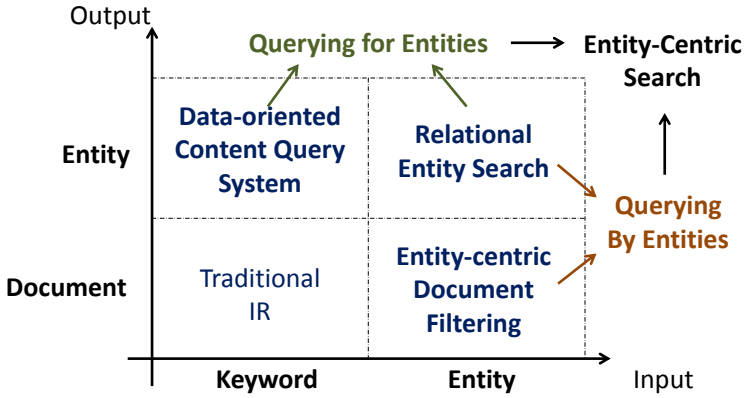


Figure 1.1: Entity-centric search.

To overcome such limitations, in my thesis, I propose to study a new entity-centric search framework, which, moving beyond the traditional page view of the Web data, is capable of supporting various entity-centric search operations.

Involving the concept of entity, an entity-centric search operation usually boils down to finding more information about some entities that are already known (*i.e.*, having entities in input queries), or exploring unknown entities that satisfy certain information need (*i.e.*, with entities in output results). To define entity-centric search, Figure 1.1 draws four quadrants to categorize search operations based on their input and output – in contrast with tractional IR, which has keywords as queries and documents as results, entity-centric search covers the other three quadrants where entities are involved. Specifically, according to whether entities appear in the input or the output, we further categorize entity-centric search operations into two categories: *querying by entities* and *querying for entities*.

In *querying by entities*, users have a clear target entity in mind, and wish to collect relevant information about the entity from the Web. For example, many people have their favorite celebrities such as movie stars and security analysts, and are interested in tracking celebrities' activities everyday; for business people, they are interested in collecting useful user experiences for their products, which is critically helpful for future quality improvement. In such examples, entities are involved in the input, and users expect to find out some entity-related information, usually represented in the form of documents.

In contrast with querying by entities with entities in the input, in *querying for entities*, users expect to retrieve some particular entities in the returned result. For example, when students are applying to PhD programs, they would need to know which universities are the top in their interested fields; when a PhD student is surveying a research topic, he/she might want to find out who are the leading researchers in that area. In these scenarios, people are looking for some particular entities (*e.g.*, universities, professors) that satisfy some information needs.

There also exist scenarios where the concept of entity appears in both input queries and output results, *i.e.*, belonging to querying by entities and for entities simultaneously. For example, when a customer is unsatisfied with a newly bought iPad, he/she would search the phone number of Apple's customer service for complaints. In such an example, the input entity would be "Apple's customer service," and the target entity is a phone number. Similar examples include finding the CEO of Amazon, the treatment for anxiety disorders, *etc.*. In these scenarios, people are interested in finding entities that match some particular relations with given entities.

To conduct a thorough research, as demonstrated in Figure 1.1, I propose to study three essential and challenging research problems which cover three different quadrants of entity-centric search operations,

specifically, entity-centric document filtering (querying by entities), data-oriented content query system (querying for entities) and relational entity search (querying by and for entities).

Entity-Centric Document Filtering [80, 81]. In querying by entities, I study the entity-centric document filtering problem, which aims at building a document filter to automatically recognize relevant documents for given entities. To help the computer better understand the targeted entities, I propose to use their identification pages (*e.g.*, a Wikipedia page, a personal homepage, *etc.*) as input. Such identification pages provide rich information about the entity; however, at the same time, they also bring a lot of noise. In order to prevent the noisy information from hindering the ranking accuracy, the entity-centric document filtering problem boils down to identifying the most essential information which best characterizes the target entity from its identification page.

As my first solution [80], I study how to learn the importance of keywords from entities' identification pages given labeled documents as training data. Based on the insight that keywords sharing some similar properties should have similar importance for their respective entities, I propose a novel concept of meta-feature to map keywords from different entities, and develop two different models – LinearMapping and BoostMapping.

I further improve the solution [81]. Observing that the contribution of a keyword to document relevance depends on not only its importance for the target entity, but also its occurrence in the document, I propose the idea of feature decoupling, which suggests defining not only meta-features over keywords and entities, but also intra-features beyond keywords and documents. To compute the document relevance based on the decoupled features, I propose a novel graphical model, Tree-Structured Redistricted Boltzmann Machine. Experiments on different datasets confirm the effectiveness of our proposed model, which show significant improvement compared with baseline methods.

Data-Oriented Content Query System (DoCQS) [82]. In querying for entities, we are witnessing many different ad-hoc efforts for exploiting fine granularity entity information across Web text, such as Web information extraction [14, 23, 52], typed-entity search [17, 13, 19], question answering [11, 47, 75], *etc.*. There is a pressing need to distill their essential capabilities. Therefore, I propose to study the concept of Data-Oriented Content Query System to support "content querying" for finding various entity data over the Web.

We observe that, as their functional requirements, all such entity-targeted applications can be distilled into three key capabilities of supporting: 1) extensible data types, 2) flexible contextual Patterns and 3) customize scoring. Such capabilities call for a conceptually relational model, upon which

I design a powerful *Content Query Language* (CQL). For efficient processing, I design novel index structures and query processing algorithms. I evaluate our proposal over two concrete domains of realistic Web corpora, demonstrating that our query language is rather flexible and expressive, and our query processing is efficient with reasonable index overhead.

Relational Entity Search [83]. In querying by and for entities, I study the relational entity search problem, to automatically learn relation-specific rankers for entity ranking. Nowadays, there is abundance of relational data stored in various knowledge bases, *e.g.*, Wikipedia, Freebase, which provides rich training data for learning different relation-specific rankers.

To realize effective ranking, I essentially exploit the redundancy of the Web – if a relation is repeatedly mentioned in different snippets of an entity, the entity is more likely to matched the desired relation. However, such redundancy is usually noisy, as we only have labels defined on the entity level, and obtaining detailed labels for each snippet is manually impractical; furthermore, we require that the ranking function should also be online executable. As our solution, We develop Pattern-based Filter Network (PFNet), a novel probabilistic graphical model. To balance the accuracy and efficiency requirements, PFNet selects a limited size of indicative patterns to filter noisy snippets, and inverted indexes are utilized to retrieve required features. Experiments on the large scale CuleWeb09 data set for six different relations confirm the effectiveness of the proposed PFNet model, which outperforms five state-of-the-art relational entity ranking methods.

In the rest of the dissertation is organized as follows. Chapter 2 and 3 present my work on entity-centric document filtering. Chapter 4 introduces my work on data-oriented content query system. Chapter 5 describes my work on relational entity search. Finally, I will make a conclusion in Chapter 6.

Chapter 2

Query By Entities: Entity-Centric Document Filtering

2.1 Introduction

The immense scale of the Web has rendered itself as a huge repository system storing information about various types of entities (*e.g.*, persons, locations, companies, *etc.*). Much of information sought on the Web nowadays is about retrieving information related to a particular entity. For example, consider the following scenarios:

Scenario 1: Knowledge Base Acceleration [1]. Knowledge bases such as Wikipedia and Freebase grow very quickly on the Web; however, it is a heavy burden for editors to digest a huge amount of new information every day and keep knowledge bases up to date. To reduce Wikipedia editors' burden, NIST proposed the TREC Knowledge Base Acceleration (TREC-KBA) problem to automatically recommend relevant documents for a Wikipedia entity based on the content of its Wikipedia page.

Scenario 2: Business Intelligence. For a company, automatically collecting useful user experiences about its product entities is helpful for future quality improvement. Moreover, from a crisis management perspective, constantly monitoring user opinions on the Web also helps the company detect potential or emergent crises in a timely manner.

Scenario 3: Celebrity Tracking. Nowadays, the success of micro-blogs is largely due to their courtship of celebrity users, as a lot of people are interested in tracking the activities of their favorite celebrities (such as movie stars and securities analysts) on micro-blogs every day. For the same reason, it would be promising to design a system that can automatically track interesting event updates of celebrities from other Web data.

In these scenarios, people are interested in an *entity-centric document filtering* system which can automatically identify relevant documents from a large text corpus (*e.g.*, the Web) or a continuous data stream (*e.g.*, news feeds) for an entity (*e.g.*, products, celebrities). In this chapter, we will study such an *entity-centric document filtering* task, and in particular address its central issue of the *learning to filter* problem.

As the input, the entity-centric document filtering system needs knowledge about the query entity that we are keen on – only an entity name might not be sufficient, since it provides too limited information to

help determine document relevance and there exist ambiguous entities sharing the same name. Nowadays, most of entities have their *identification pages* accessible on the Web (*e.g.*, Wikipedia pages for well-known entities, Amazon pages for products, *etc.*), which usually contain rich information covering different aspects of the entities. In order to better characterize the target entity, we will feed its identification page as the input of the system. Based on such rich information, the system has to tackle with *how to justify if a document is relevant to a specific entity described by an identification page*, *i.e.*, to realize *entity-centric document filtering*. We emphasize that, although the input is an entity identification page, relevance here represents the sense of *entity-relevance*, which measures whether the document contains related information about the entity, rather than whether it repeats similar content of the entity identification page.

As a manually crafted formula might not well characterize the entity-relevance of a document, in this chapter, we study the *learning to filter* problem to realize entity-centric document filtering in a learning framework. In the offline training phase, we are given some training entities (*e.g.*, "Bill Gates"), each of which is represented by an identification page (*e.g.*, the Wikipedia page of "Bill Gates") and associated with a set of documents labeled as relevant/irrelevant. Our goal is to learn an entity-centric document filter, such that at the online query time, given a new *query entity* (*e.g.*, "Michael Jordan") represented by its own identification page, the filter can correctly identify its relevant documents.

With entity identification pages as the input and measuring entity-relevance of documents as the goal, our problem needs a new methodology as the solution. Most existing document retrieval techniques (*e.g.*, BM25 [64], learning to rank [15]) deal with short keyword queries and can not handle our problem whose input is a long entity identification page. With a document as input, one straightforward solution is to define the document relevance as the cosine similarity between the entity identification page and the target document. However, such an idea can not well characterize the semantics of entity-relevance, because for most of time, an entity-relevant document only discusses one aspect of the entity or reports a latest event not recorded in the identification page, and thus, would not have high similarity.

To justify the entity-relevance of a document, a better solution is to check if the document mentions about the most basic information that is commonly used to identify the entity in its relevant documents. For example, for a document relevant to "Bill Gates," with high probability, it will mention important keywords such as "Microsoft," "Seattle," "philanthropist" which hint its entity-relevance to "Bill Gates." We believe that an entity identification page, as a comprehensive summarization of an entity, should contain such important keywords. Since entity identification pages are usually long and also contain other less important keywords, in entity-centric document filtering, we are mandated to *discriminate the importance of keywords in an identification page, and use those important keywords to score the entity-relevance of a*

document.

However, although the problem boils down to the learning of keyword importance, it is non-trivial to generalize keyword importance from training entities to new unseen entities. We can easily learn the keyword importance for training entities (*e.g.*, "Microsoft" is an important keyword for "Bill Gates") by treating each keyword as a feature and building a document classifier [33] to learn the keyword weighting; however, for a new entity such as "Michael Jordan," the learned weighting is not applicable as keyword "Microsoft" is no longer important. Towards learning a model that is generalizable to new entities, we abstract our problem as a *transfer learning* problem, and study how to appropriately transfer the keyword importance learned from training entities to new query entities which do not have labeled documents.

As the *key contribution* of this work, we propose a novel concept of *meta-feature* to realize transfer learning. Intuitively, if two keywords from two entities (*e.g.*, "Microsoft" for "Bill Gates" and "ChicagoBulls" for "Michael Jordan") share some similar properties (*e.g.*, both "Microsoft" and "ChicagoBulls" are organization names, and mentioned a lot in their identification pages), these two keywords should have similar importance for their respective entities. We name such properties of keyword features as *meta-features*, based on which the keyword importance transfer is enabled. For example, although we do not have labeled documents for "Michael Jordan," by mapping "Microsoft" and "ChicagoBulls" based on their meta-features, we can infer that "ChicagoBulls" is important for "Michael Jordan," given the training result showing that "Microsoft" is important for "Bill Gates." We formalize such an intuition as the *meta-feature-based feature mapping principle*.

To support the idea of feature mapping, the designed meta-features should be related with the importance of keyword features. We not only use traditional keyword importance measures such as TF-IDF, but also design many meta-features characterizing how a keyword appear in a *structured* entity-identification page. Such structural information is crucial for determining keyword importance, *e.g.*, meta-feature *InfoxBoxTF*, denoting the frequency of a keyword in the infobox of a Wikipedia page, is designed because keywords mentioned in the infobox tend to be more important.

Based on the designed meta-features, we propose two different models, *LinearMapping* and *BoostMapping*, to realize the idea of feature mapping. Assuming the keyword importance as a simple linear function of meta-features, LinearMapping could be realized by a standard classifier. For BoostMapping, it clusters keyword based on their meta-features (*e.g.*, by clustering keyword "ChicagoBulls" and "Microsoft" together), and assume that keywords from the same cluster share the same importance. Without relying on the linear assumption, BoostMapping has stronger representation power compared with LinearMapping.

In the design of BoostMapping, we tackle with a challenging *distantly supervised clustering* problem –

as it is not possible to guarantee the designed meta-features are all related with entity-centric document filtering, the clustering process might be interfered by the existence of irrelevant meta-features. As we do not have labels on the keyword level indicating how keywords should be clustered, we will study how to cluster keywords guided distantly by *labels on the document level*, *i.e.*, realizing *distantly supervised clustering*, to generate useful keyword clusters in BoostMapping.

In the experiments, we performed our evaluation on three different datasets: TREC-KBA, Product and MilQuery, to validate the effectiveness of LinearMapping and BoostMapping on different types of entity identification pages (Wikipedia pages for TREC-KBA and MilQuery, Amazon product pages for Product), and on different types of entities (entities are of similar types in TREC-KBA and Product, and of general types in MilQuery). We compared our models with four state-of-the-art baselines, and observed consistent improvement.

2.2 Related Work

In terms of application, different from most information retrieval applications [64, 15] whose queries are a small number of keywords, the input of entity-centric document filtering is a document characterizing the target entity. With an entity identification page as input, our problem is closely related to [41, 76, 74], which measures document relevance with respect to a long keyword query [41] or a query document [76, 74]. Kumaran et al. [41] propose to enumerate all possible short sub-queries of the input long query, and learn a query quality predictor to select the best one. Such an approach is not applicable for our problem because there are too many sub-queries when the input is a long document. Weng et al. [74] propose to calculate the relevance of a document based on its topical similarity to the query document (*e.g.*, relevant if both documents are about politics); different from their work, our task aims at retrieving entity-relevant documents rather than ones discussing about similar general topics. Yang et al. [76] propose to extract important phrases from the query document, and use the phrases as keyword queries. Although they also discriminate the importance of keywords in the query document, their solutions rely on a manually-crafted scoring function, which can hardly characterize the semantics of entity-relevance. In contrast, our work will study how to automatically learn the criteria of entity-relevance from training data.

In terms of abstraction, without labeled documents for query entities, we view entity-centric document filtering as a transfer learning problem, to transfer keyword importance across entities. Most transfer learning works [10, 21, 78] assume the existence of domain-independent features, which have the same feature importance for different domains. For example, in sentimental classification [10], domain-independent

features could be keywords like "good" and "bad" which are strongly indicative of a positive/negative review in all domains (*e.g.*, book and electronic product reviews). Their challenges arise in the different distributions of domain-independent features in different domains and the interference of domain-dependent features. As their solutions, they propose approaches including instance re-weighting [78], feature representation transfer [21] *etc.*, to minimize the difference in domain-independent feature distributions across different domains and remove the interference of domain-dependent features. In our problem, with entity-relevance as the goal, the keyword weightings should represent their importance for a particular entity, and thus, are all entity-specific (domain-dependent). In the absence of domain-independent features that are strongly indicative of a relevant document for all entities, we propose a novel transfer learning framework relying on "meta-features" to bridge domain-dependent features.

In terms of feature mapping, our work is closely related with [10] which proposes Structural Correspondence Learning (SCL) to bridge domain-dependent keywords based on domain-independent keywords (so-called "pivot keywords"). Their intuition is that, if two domain-dependent keywords (*e.g.*, "tedious" for book reviews and "fragile" for product reviews) have similar co-occurrence frequency with some pivot keywords (*e.g.*, "bad"), these two keywords should represent similar semantics (*i.e.*, indicative of a negative review) in their respective domains. As their solution, for each pivot keyword in each domain, they train one pivot predictor predicting how likely the pivot keyword will appear given other domain-dependent features, and use the output of the pivot predictor as a new feature representation, which implicitly maps domain-dependent features. For the aforementioned reason that we are lack of such domain-independent features (pivot keywords), SCL is not applicable to our problem; however, we believe that our framework is a more general solution for feature mapping, which can solve the sentimental classification problem by defining meta-features measuring how each domain-dependent keyword co-occur (*e.g.*, mutual information) with each pivot keyword. We will verify such an idea in the future work, as it is out of the scope of this chapter.

In the design of BoostMapping, we study the distantly supervised clustering problem, to generate useful keyword clusters for predicting document relevance. Similar motivation has been studied by [9], which proposed a supervised topic model called sLDA to infer useful topic clusters that are predictive of the document labels. However, as the key difference, sLDA clusters keywords according to their co-occurrences in documents, while we aim at clustering keywords based on their meta-features. To the best of our knowledge, such a meta-feature-oriented distantly supervised clustering problem has not been studied in exiting research works.

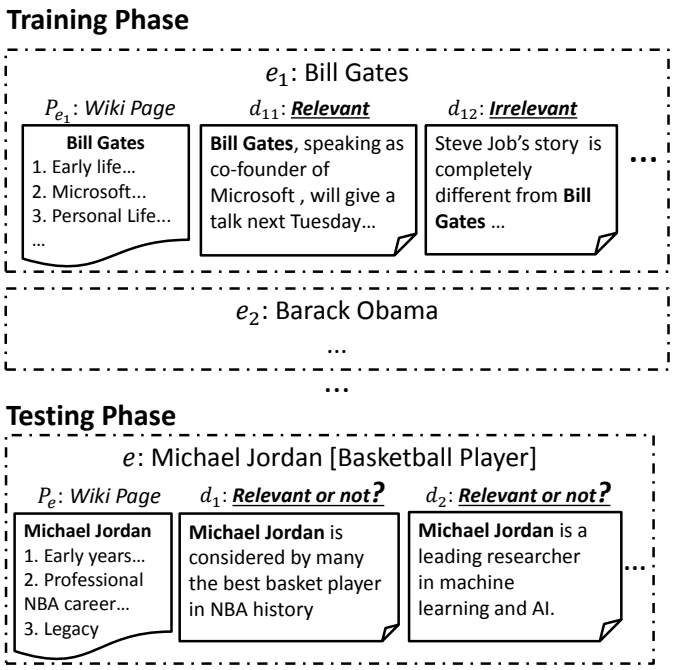


Figure 2.1: Entity-centric document filtering.

2.3 Entity-Centric Filtering

This section formally defines the entity-centric document filtering problem. We abstract our problem as a transfer learning problem, and propose a novel concept of *meta-feature* to bridge keyword features across different entities.

2.3.1 Problem & Framework

Problem: Learning to Filter. To realize *entity-centric document filtering*, we shall build a filter $\mathcal{F}$ to determine, for an entity e, the relevance of a document d, *i.e.*, if d provides information about e. Specifically, given an entity e (*e.g.*, "Michael Jordan [Basketball Player]") as described by its *identification page* $\mathcal{P}_e$ (*e.g.*, the Wikipedia page of e), if d is *relevant* to e, *i.e.*, $\mathcal{F}(d \mid e, \mathcal{P}_e) = 1$, or *irrelevant*, *i.e.*, $\mathcal{F}(d \mid e, \mathcal{P}_e) = -1$, *e.g.*, as Figure 2.1 shows, d_1 is relevant but d_2 is not, since it refers to a computer scientist.

Our objective is to automatically learn such a filter, *i.e.*, *learning to filter*. In the *training phase*, we are given a set of *training entities* $\mathbf{e} = \{e_1, e_2, ..., e_N\}$ (*e.g.*, "Bill Gates" and "Barack Obama" in Figure 2.1), each of which is described by an identification page $\mathcal{P}_{e_i}$, and associated with a set of example documents $\mathbf{d_i} = \{d_{i1}, d_{i2}, ..., d_{i|\mathbf{d_i}|}\}$ with labels $y_i = \{y_{i1}, y_{i2}, ..., y_{i|\mathbf{d_i}|}\}$, where $y_{ij} = +1$ (or -1) indicates that d_{ij} is relevant (or irrelevant) to e_i. *E.g.*, in Figure 2.1, for entity "Bill Gates," d_{11} is labeled as relevant since it describes Bill Gate's activity, while d_{12} is irrelevant because it is mainly about Steve Jobs. Based on the training data, our goal is to learn a document filter $\mathcal{F}$, such that, in the *testing phase* (or the "query time"),

$\mathcal{F}$ could correctly predict document relevance for a new entity e (*e.g.*, "Michael Jordan") which does not appear in training entities, *i.e.*, $e \notin \mathbf{e}$.

Framework. As discussed in Section 2.1, the nature of entity relevance mandates us to check if it mentions about the basic information of the entity. To fulfill such a goal, we define the relevance of a document as the summation of the contribution of its keywords, thus, a document that mentions more important keywords, which represents the basic information of the query entity, tends to be more relevant. Formally, if we define $\mathcal{V}^{(\mathcal{P}_e)}$ as a set of keywords mentioned in the identification page $\mathcal{P}_e$, $weight(w, e)$ to measure the importance of keyword w, $tf(w, d)$ to denote the frequency of keyword w in document d and $len(d)$ to denote the length of document d, the document relevance $rel(d \mid e, \mathcal{P}_e)$ is defined as

$$rel(d \mid e, \mathcal{P}_e) = \frac{\sum_{w \in \mathcal{V}^{(\mathcal{P}_e)}} weight(w, e) * tf(w, d)}{len(d)} \tag{2.1}$$

In Eq. 2.1, the relevance score is normalized by $len(d)$ to avoid the bias towards long documents.

We note that Eq. 2.1 is both natural and widely adopted. Intuitively, to justify the relevance of a document, we need to check how the document is written, or more specifically, which keywords are used in the document. Such an idea of measuring the document relevance as the summation of its keyword contribution has been widely adopted in various IR applications. For example, in document classification [33], the document relevance takes the same form of Eq. 2.1 with the keyword weighting measuring the relevance of keyword w to a category (*e.g.*, "economy").

With the simple relevance modeling, we can further concretize the filter $\mathcal{F}$ to learn. Since $\mathcal{F}$ is to judge if a document d is sufficiently relevant to an entity e identified by $\mathcal{P}_e$, we can simply base the decision on the relevance value,

$$\mathcal{F}(d \mid e, \mathcal{P}_e) = \begin{cases} +1 & \text{if } rel(d \mid e, \mathcal{P}_e) \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.2}$$

Overall, we realize the entity-centric filtering task with the following framework. The framework, as a learning approach, consists of two stages: At training time, it needs to learn the *weight* function, which recognizes the importance of each keyword w for a new entity. With such a function learned, at testing time, we simply apply *weight* to realize the filtering function.

Training Phase:

Given: Training entities $\mathbf{e} = \{e_1, \ldots, e_N\}$, each entity e_i
is associated with an identification page $\mathcal{P}_{e_i}$;

Doc. $\mathbf{d_i} = \{d_{i1}, \ldots, d_{i|\mathbf{d_i}|}\}$; labels $\mathbf{y_i} = \{y_{i1}, ..., y_{i|\mathbf{d_i}|}\}$.

Output: $weight(w, e)$.

Testing Phase:

Given: Query entities e; Identification page p. Candidate document d;

Output: $\mathcal{F}(d \mid e, p)$ by Eq. 2.2.

2.3.2 Challenge: Learning across Entities

To learn the relevance function in Eq. 2.1, with $tf(w, d)$ and $len(d)$ both given, the entity-centric document filtering problem boils down to determining the value of $weight(w, e)$, *i.e.*, the importance of keyword w for entity e.

To clarify the challenge of our problem, we start with a simpler case which assumes that document training labels are available for query entities, *i.e.*, the original task studied in TREC-KBA which assumes $e \in \mathbf{e}$. Such a case could be simply solved by traditional document classification techniques [33], by learning one document filter for each query entity based on its labeled documents. Taking entity "Michael Jordan" as example, referring to [33], we define each $\frac{tf(w,d)}{len(d)}$ as one feature corresponding to keyword w (*e.g.*, "ChicagoBull"), and learn feature weighting $weight(w, e)$ by SVM; in the query time, $weight(w, e)$ will be applied to predict the relevance of a new document for "Michael Jordan".

However, since we believe that the requirement of document labels for every entity is impracticable and assume that the query entity does not have labeled documents, traditional document classification techniques no longer work. As the same keyword (*e.g.*, "Microsoft") will have very different importance for different entities (*e.g.*, "Bill Gates" and "Michael Jordan"), the keyword weighting learned from one entity could not be generalized to others. To tackle with this problem, we will study how to *appropriately transfer the keyword importance across different entities*, which will be the core challenge in this work.

The idea of transferring feature weightings across entities naturally connects our problem to *transfer learning* [10, 21, 78, 57]. Different from most machine learning problems which assume that the training and testing data must be in the same feature space with generalizable feature weightings, *i.e.*, in the same domain, transfer learning studies, once the same-domain assumption does not hold, how to transfer knowledge across different domains. We abstract entity-centric document filtering as a transfer learning problem, with *each entity corresponding to one domain*. For different entities, the domain difference is reflected by the fact that the keyword features (*e.g.*, "microsoft") are *domain-dependent*, which have different importance for different

entities/domains (*e.g.*, "Bill Gates" and "Michael Jordan").

To tackle with the existence of domain-dependent features, existing transfer learning works either filter out such domain-dependent features and rely on domain-independent features [55, 24], or try to learn a new transferable feature representation for different domains [10]. For example, [10] proposes to use the output of "pivot predictors" as a new feature presentation which relates domain-dependent keywords from different domains. However, all such works heavily rely on the existence of domain-independent features, while in entity-centric document filtering, as our goal is to identify the entity-relevance of a document, the keyword weightings should represent the importance of keywords for a particular entity, and thus, are all entity-specific. As there are no domain-independent keyword features that are commonly predictive of entity-relevance for all entities, the abstraction of entity-centric document filtering raises a novel transfer learning problem, *i.e.*, *how to realize knowledge transfer in the absence of domain-independent features.*

2.3.3 Key Insight: Features of Features

In order to realize transfer learning based on only domain-dependent features, we propose a novel concept of *meta-feature*. As our key insight, although a domain-dependent feature has different weightings in different domains, if the weighting is related with some domain-independent properties, we can use such properties to bridge domain-dependent features. For example, in entity-centric document filtering, given keyword "Microsoft" from entity "Bill Gates", and "ChicagoBulls" from "Michael Jordan", as both keywords are organization names mentioned a lot in their entity identification pages, intuitively they should have similar importance for their respective entities. As these properties are defined on keyword features w characterizing their potential importance for entity e, we name such "features of features" as *meta-features*, denoted by $\mathbf{f}(w, e) := \langle f_1(w, e), ..., f_K(w, e)\rangle$ where $f_k(w, e)$ indicates the k-th meta-feature function of keyword w with respect to e. Such an insight could be summarized as the *Meta-Feature-based Feature Mapping principle*, described as follows,

Principle 1 (Meta-Feature-based Feature Mapping): Given two keyword-entity pairs $\langle w_1, e_1\rangle$ and $\langle w_2, e_2\rangle$, if the two meta-feature vectors $\mathbf{f}(w_1, e_1)$ and $\mathbf{f}(w_2, e_2)$ are similar, $weight(w_1, e_1)$ is similar to $weight(w_2, e_2)$. ∎

Principle 1 proposes to map keywords sharing similar meta-features together to share similar weightings, which enables transfer learning across entities. In the training phase, we learn w (*e.g.*, "Microsoft") is important for e_i (*e.g.*, "Bill Gates") with high $weight(w, e_i)$, whereas, at the query time, although we do not have labeled documents for new entity e (*e.g.*, "Michael Jordan"), we can still infer w'(*e.g.*, "ChicagoBull") is important for e by observing $\mathbf{f}(w, e_i) \approx \mathbf{f}(w', e)$.

Name	Explanation
IdPageIDF	Inverse document frequency of *w* in identification pages
IdPageTF	Term frequency of *w* in the identification page of *e*
DocIDF	Inverse document frequency of *w* in document corpus
EntityIDF	Inverse document frequency of *w* in articles containing *e*
IdPagePos	First position of *w* in the identification page of *e*
IsNoun(Verb/ Adj/Adv)	If *w* is a noun word (verb word/adjective word/adverb word)
InName	If *w* is included in entity name.
InfoBoxTF	Term Frequency of *w* in *the InfoBox*
OpenParaTF	Term Frequency of *w* in the opening paragraph
RefTF	Term Frequency of *w* in the reference.
AnchorEntity	Term Frequency of *w* in the anchor text of entities.

Figure 2.2: Meta-features $\mathbf{f}(w, e)$ for TREC-KBA.

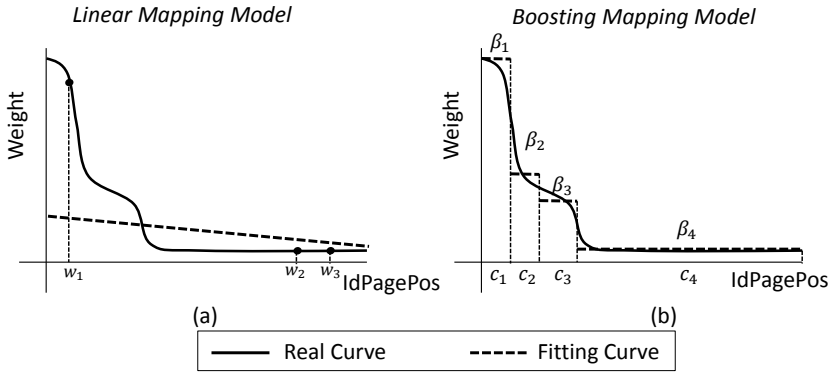


Figure 2.3: $weight(w, e)$ curves in two approaches.

Figure 2.2 lists all meta-features we design for the TREC-KBA task, which characterize the potential importance of keyword w for entity e. For example, $IdPagePos(w, e)$ denotes the first position of keyword w in the identification page of e, and for most of time, keywords mentioned earlier in the identification page (*e.g.*, a Wikipedia article) tend to more important. We also incorporate the structural information of the entity-identification page in the design of meta-feature, *e.g.*, a keyword mentioned in the info-box with non-zero $InfoBoxTF(w, e)$ is likely to be important.

2.4 Meta-Feature Based Feature Mapping

Based on the insight proposed in Section 2.3.2, our objective boils down to mapping keyword features from different entities by their meta-features. In this section, we propose and contrast two feature mapping models – LinearMapping and BoostMapping – corresponding to defining $weight(w, e)$ as a continuous function and a discrete function respectively. We compare the pros and cons of these two models at the end of the section.

2.4.1 Continuous Linear Mapping Model

To model Principle 1, one simple idea is to define $weight(w, e)$ as a continuous function of the meta-feature vector of $\langle w, e\rangle$ – the continuous property, which indicates that the function will produce similar output with similar input, naturally fulfills the requirement of Principle 1, *i.e.*, realizing the idea of feature mapping.

To facilitate the learning of parameters, we study the *linear mapping model* (LinearMapping), which assumes the simplest linear form of $weight(w, e)$, formally defined as follows,

$$weight(w, e) = \sum_{k=1}^{K} \alpha_k * f_k(w, e) \tag{2.3}$$

where α_k measures the importance of meta-feature $f_k(w, e)$. By inserting Eq. 2.3 into Eq. 2.1, we obtain

$$\begin{aligned} rel(d \mid e, p) &= \sum_{k=1}^{K} \alpha_k * \frac{\sum_{w \in \mathcal{V}} f_k(w, e) tf(w, d)}{len(d)} \\ &= \sum_{k=1}^{K} \alpha_k * l_k(d, e) \end{aligned} \tag{2.4}$$

where $l_k(d, e) := \frac{\sum_{w \in \mathcal{V}} f_k(w,e) tf(w,d)}{len(d)}$ is defined as the summation of one meta-feature of all keywords.

The linear assumption of $weight(w, e)$ largely facilitates the learning of parameters. We can view $\langle l_1(d, e), ..., l_K(d, e)\rangle$ as a document feature vector generated by LinearMapping for document d, and the problem is transformed into a standard classification problem. We can apply any standard classification technique such as SVM, logistic regression to learn α_k according to document labels.

Figure 2.3(a) intuitively shows how LinearMapping learns a linear curve (*i.e.*, the dash curve) to fit the real curve (*i.e.*, the solid curve, taking an L-shape because keywords appearing at the top of an Wikipedia page are more important). In addition to the continuous linear function, $weight(w, e)$ could also be defined as a discrete function, plotted in Figure 2.3(b), which will be discussed in the following section.

2.4.2 Discrete Boosting Mapping Model

In this section, we propose a novel boosting mapping (BoostMapping) model which defines $weight(w, e)$ as a discrete function. Such a design is motivated from the idea of constructing keyword clusters to realize feature mapping. In BoostMapping, we will study how to guide the clustering process such that the generated clusters are useful for document filtering, *i.e.*, the distantly supervised clustering problem.

Feature Mapping based on Keyword Clusters

As the key idea, we cluster keywords based on their meta-features, and assume that keywords belonging to the same cluster contribute the same $weight(w, e)$ to document relevance. For example, by clustering keyword "Microsoft" for "Bill Gates" and "ChicagoBull" for "Michael Jordan" together, we can learn the keyword weighting of "Microsoft" for "Bill Gates" from the training data, and apply it to "ChicagoBull" for "Michael Jordan" at the query time.

To formalize such an idea, we define $\mathcal{L} = \{\langle w, e\rangle | e \in \mathbf{e}, w \in \mathcal{V}^{(\mathcal{P}_e)}\}$ containing all keyword-entity pairs, and $\mathbf{c} = \{c_1, ..., c_M\}$ as a set of keyword clusters, where each keyword cluster $c_m \subseteq \mathcal{L}$ contains keyword-entity pairs sharing similar meta-features. The keyword importance $weight(w, e)$ depends on which clusters $\langle w, e\rangle$ belongs to, defined as,

$$weight(w, e) = \sum_{m=1}^{M} \beta_m I_{c_m}(w, e) \tag{2.5}$$

where $I_{c_m}(w, e) \in \{0, 1\}$ is an indicator function denoting whether $\langle w, e\rangle$ belongs to keyword cluster c_m, and β_m measures the importance of cluster c_m. By inserting Eq. 2.5 into Eq. 2.1, we obtain

$$\begin{aligned} rel(d \mid e, \mathcal{P}_e) &= \sum_{m=1}^{M} \beta_m \frac{\sum_{w \in \mathcal{V}^{(\mathcal{P}_e)}} I_{c_m}(w, e) tf(w, d)}{len(d)} \\ &= \sum_{m=1}^{M} \beta_m h_m(d, e) \end{aligned} \tag{2.6}$$

Similar to LinearMapping, $h_m(d, e) = \frac{\sum_{w \in \mathcal{V}^{(\mathcal{P}_e)}} I_{c_m}(w,e) tf(w,d)}{len(d)}$ could be viewed as a document feature generated by BoostMapping. Eq. 2.5 actually defines $weight(w, e)$ as a discrete step function, as plotted in Figure 2.3(b). Each interval represents a keyword cluster across entities. Intuitively from the figure, with a good design of keyword clusters $\mathbf{c}$, such a discrete function should better fit the underlying real curve of arbitrary shape compared with LinearMapping.

Distantly Supervised Clustering

The idea of keyword-clustering-based feature mapping raises a new challenge for us – as we do not have keyword labels indicating how keywords should be clustered, if we adapt traditional clustering algorithms (*e.g.*, Guassian Mixtures, K-Means) to cluster keywords based on their meta-features, the generated keyword clusters might be misleading for entity-centric document filtering. For example, a meta-feature which measures the length of a keyword might tempt traditional clustering algorithms to construct a keyword cluster

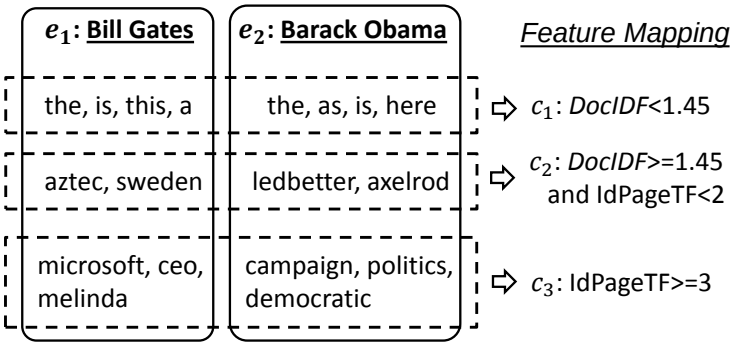


Figure 2.4: Example of feature mappings.

containing only long keywords, which is not very helpful for identifying the keyword importance. Ideally, the process of keyword clustering should be guided by document labels, to generate meaningful keyword clusters for document relevance prediction.

We abstract such a problem of finding meaningful keyword mappings (or clusters) $\mathbf{c}$ for predicting $\mathbf{y}$ as the *distantly supervised clustering* problem. It is *distantly supervised*, because, without keyword labels indicating how keywords should be clustered, the only supervision comes from document labels on a different level. Formally, in the training phase, given a set of example documents $\mathbf{d} = \{d_1, d_2, ..., d_{|\mathbf{d}|}\}$ with labels $\mathbf{y} = \{y_1, y_2, ..., y_{|\mathbf{d}|}\}$, our goal is to cluster $\mathcal{L} = \{\langle w, e\rangle | e \in \mathbf{e}, w \in \mathcal{P}_e\}$ into a set of feature mappinps $\mathbf{c} = \{c_1, ..., c_M\}$, such that, first, keywords in the same c_m should share similar meta-features $\mathbf{f}(w, e)$, and second, $\mathbf{c}$ should be useful for predicting $\mathbf{y}$. In the testing phase, we will map new keywords to c_i according to the same clustering criteria, and predict the document relevance based on Eq. 2.6.

Boosting Mapping Model

In this section, we propose a novel boosting mapping model (BoostMapping) to tackle with the distantly supervised clustering problem.

As the key idea, in order to obtain useful keyword clusters $\mathbf{c}$ for predicting $\mathbf{y}$, we will jointly optimize $\mathbf{c}$ together with parameters $\boldsymbol{\beta}$ to minimize the prediction error, rather than isolating the clustering process from the optimization. Much like other machine learning models, the objective of BoostMapping is to minimize the loss between document labels $\mathbf{y}$ and the document relevance predicted by Eq. 2.6, with respect to some loss function,

$$\underset{\mathbf{c},\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{d,e} loss[y, \sum_{m=1}^{M} \beta_m \frac{\sum_{w \in \mathcal{V}(\mathcal{P}_e)} I_{c_m}(w, e) tf(w, d)}{len(d)}] \tag{2.7}$$

In Eq. 2.7, different selections of loss function will largely affect the difficulty of optimization. For example, if we use the loss function of logistic regression or SVM, we have to jointly construct all clusters $\mathbf{c}$ to optimize

the resulting objective function, which is complicated and intractable.

In order to derive a *tractable* objective function, BoostMapping adapts the idea of AdaBoost [27] as the solution. The main idea of AdaBoost is to iteratively learn a list of weak learners such that the subsequent learners are tweaked in favor of those instances misclassified by previous learners. Following the same idea, we view $h_m(d,e) = \frac{\sum_{w \in \mathcal{V}(\mathcal{P}_e)} I_{c_m}(w,e) tf(w,d)}{len(d)}$ in Eq. 2.6 as a weak learner corresponding to one cluster c_m, and thus the goal of finding predictive keyword clusters c_m turns out to be the process of seeking the best weak learner h_m. As the key advantage of such a boosting idea, the objective function of cluster construction becomes very simple which learns only one cluster in each iteration.

Formally, we use the same exponential loss function with AdaBoost, given by

$$\sum_{i=1}^{N} \sum_{j=1}^{|\mathbf{d_i}|} \exp[-y_{ij} * (\sum_{m=1}^{M} \beta_m h_m(d_{ij}, e_i))] \tag{2.8}$$

To minimize Eq. 2.8, BoostMapping defines an importance distribution $D(d,e)$ over all entity-document pair $\langle d, e \rangle$. Initially, the distribution is set to be uniform with $D_1(d,e) = \frac{1}{|\mathcal{L}|}$, where $|\mathcal{L}|$ is the total number of entity-document pairs. In the m-th iteration, we search the optimal keyword cluster c_m (the detailed design of c_m will be discussed later) whose corresponding weak learner h_m achieves the lowest error rate with respect to distribution $D_m(d,e)$,

$$\underset{h_m}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j=1}^{|\mathbf{d_i}|} D_m(d_{ij}, e_i) \delta[y_{ij}, h_m(d_{ij}, e_i)] \tag{2.9}$$

where $\delta(y,h)$ is an function indicating if the output of $h(d,e)$ is inconsistent with document label y, defined as,

$$\delta(y,h) = \begin{cases} 1 & y = -1 \text{ and } h > 0 \\ & \text{or } y = 1 \text{ and } h \leq 0 \\ -1 & \text{elsewise} \end{cases} \tag{2.10}$$

After the best h_m is chosen, β_m could be found by a simple binary search [27]. The distribution $D_m(d,e)$ is updated by

$$\begin{aligned} & D_{m+1}(d_{ij}, e_i) \\ = \; & \frac{D_m(d_{ij}, e_i) \exp[-\beta_m y_{ij} h_m(d_{ij}, e_i)]}{\sum_{i'=1}^{N} \sum_{j'=1}^{|\mathbf{d_{i'}}|} D_m(d_{i'j'}, e_{i'}) \exp[-\beta_m y_{i'j'} h_m(d_{i'j'}, e_{i'})]} \end{aligned} \tag{2.11}$$

Such a process is repeated until M keyword clusters are collected. The property of AdaBoost theoretically guarantees that Eq. 2.7 is minimized after each iteration and will finally converge.

Based on the above optimization procedure, the only problem left is how to appropriately define clusters $\mathbf{c}$ such that the optimization of Eq. 2.9 is simple. As our solution, we define a cluster as a set of keywords satisfying a set of predicates defined over meta-features, given as follows,

$$c := \{\langle w, e\rangle | \bigwedge_{l=1}^{L} p_l(w, e)\} \tag{2.12}$$

where each predicate is a binary function measuring whether one meta-feature is greater or less than a threshold (*e.g.*, DocIDF ≤ 1.45, InfoBoxTF ≥ 1).

The predicate-based design of $\mathbf{c}$ facilitates the optimization of Eq. 2.9 – we adapt a simple greedy strategy to construct keyword cluster c_m. In particular, we enumerate all meta-features and possible thresholds (note that although a threshold can be any real number, it is enumerable because the number of meaningful thresholds is less than the number of keyword-entity pairs), and find out the predicate whose corresponding keyword clusters (*e.g.*, one containing keywords satisfying DocIDF $\leq$ 1.45) minimizes Eq. 2.9. Such a process is repeated until L predicates are collected, *e.g.*, c_m could contain keywords satisfying DocIDF $\leq$ $1.45 \wedge$ InfoBoxTF $\geq 1 \wedge$ IDPageTF ≥ 10.

2.4.3 LinearMapping versus BoostMapping

In this section, we will compare the pros and cons of LinearMapping and BoostMapping in details.

As the key advantage of LinearMapping, by assuming that the weighting function takes the linear form, it could be transformed to a standard classification problem; however, its linearity assumption also results in the weaker representative power of LinearMapping compared with BoostMapping. We can intuitively understand such a drawback by checking if the generated features $l_k(d, e)$ by LinearMapping in Eq. 2.4 are meaningful document features. Still take meta-feature *IdPagePos* as example, and assume that there are three keywords: w_1, w_2 and w_3 with *IdPagePos* 0.1, 0.9 and 1.0 (measured by percentage) respectively, and two documents of the same length: d_1 mentioning w_1 once and w_3 eight times, and d_2 mentioning w_2 nine times. Intuitively, as shown in Figure 2.3, w_1 is a very important keyword as it is mentioned at the top of the entity identification page, while w_2 and w_3 are not, and thus, d_1 should be more relevant compared with d_2; however, LinearMapping generates the same $l_k = 0.81$ for both d_1 and d_2, failing to discriminate their importance. As LinearMapping simply sums up the meta-feature of all keywords, important keywords such as w_1 might be easily diluted by noisy ones such as w_2 and w_3. Essentially, such a drawback stems from

DataSet	Num of Entities	Number of Docs	Number of Positive Docs	Identification Page	Entity Types
TREC-KBA	29	52,238	24,704	Wikipedia	person
Product	39	2,398	1,040	Amazon	product
MilQuery	143	8,208	2,342	Wikipedia	general

Figure 2.5: Dataset specification.

the linear assumptions of LinearMapping. Once the real keyword weighting curve takes some shapes (*e.g.*, the L-shape curve in Figure 2.3) that can not be well fitted by a straight line, LinearMapping will perform poorly.

Different from LinearMapping, BoostMapping can better discriminate important keywords from noisy ones in documents. First, the document features $h_m(d, e)$ generated by BoostMapping in Eq. 2.6, which measure the percentage of keywords sharing similar meta-features in d, are intuitively more meaningful than $l_k(d, e)$ measuring the summation of meta-features in LinearMapping. For the aforementioned example, BoostingMapping will learn a good cluster contain only w_1 (how to learn good clusters is discussed in Section 2.4.2 by maximizing the overall likelihood), and the corresponding $h_m(d, e)$ measuring the percentage of w_1 mentioned in d will be helpful for discriminating d_1 from d_2. Second, as shown in Figure 2.3, BoostMapping learns a discrete curve which can better fit the real curve of arbitrary shape.

However, the strong representation power of BoostMapping might also lead to a potential over-fitting problem – the formed keyword cluster might bias towards a small number of training entities, and could not be generalized to new entities. Currently, we require that the keywords in each cluster should appear in at least 20% of documents for each training entity. Such a heuristic helps filter those biased clusters and greatly relieves the over-fitting problem; while we will continue to explore more principled solutions in our future works.

2.5 Experiment

In this section, we compare the overall performance of LinearMapping and BoostMapping with four different state-of-the-art baselines to demonstrate the advantages of using meta-features for entity-centric document filtering. We further compare LinearMapping and BoostMapping on different feature sets, to better understand their differences. Finally, we verify the effect of the distantly supervised clustering technique in BoostMapping.

2.5.1 Experiment Setting

In order to demonstrate the capability of LinearMapping and BoostMapping on general entity-centric filtering tasks, the evaluation collections should cover general types of entities, as well as different types of entity identification pages. Based on such a concern, we chose three different data sets with specification shown in Figure 2.5.

1. *TREC-KBA* [1]. This dataset is built based on a stream corpus containing 134 million news and 322 million social articles. The KBA track chose 29 Wikipedia entities including living persons from different domains (*e.g.*, basketball coach "Bill Coen", professor "Jim Steyer") and a few organizations (*e.g.*, "Basic Element (company)"). For each entity, $100 \sim 3000$ candidate documents are collected and labeled as garbage, neutral, relevant or central. Following the same procedure in [26], we got binary labels by viewing central and relevant documents as positive, and others negative.

2. *Product.* To demonstrate the capacity of algorithms on different types of entity identification pages, this dataset chose electronic products as query entities (*e.g.*, TV, laptop, cellphone, *etc.*) identified by their Amazon product pages. We used the same stream corpus in the TREC-KBA dataset to collect candidate documents for each entity, and manually labeled each document as relevant/irrelevant. The Product dataset uses similar meta-feature design with TREC-KBA in Figure 2.2, by replacing Wikipedia-specific meta-features with ones designed for Amazon pages, *e.g.*, $ReviewTF(w, e)$ denoting how many times w appears in the review section of e's Amazon page.

3. *MilQuery.* Different from the above datasets where almost all entities are person entities or electronic products, this dataset covers entities of different types (*e.g.*, yahoo, yogurt, tetanus vaccine, *etc.*). To build this dataset, we used the 2009 Million Query Track collection, and chose a subset of entity queries – ones that have Wikipedia pages – together with their labeled documents. The documents in the original corpus has three levels of relevance: very relevant, relevant and irrelevant, while we view very relevant and relevant as positive, and irrelevant as negative. Since Wikipedia articles are used to describe entities, MilQuery uses the same meta-feature design as the TREC-KBA dataset.

To confirm the confidence of the comparison, we used 5-fold cross validation by dividing all entities into five sets, chose 4 entity sets together with their associated documents for training, and the remaining entity set as queries for testing. Thus, each entity would not be included in the training corpus and the testing corpus at the same time.

In our experiments, we used the implementation of SVMLight [34] to learn feature weightings for LinearMapping and transform continuous score to binary output. To deal with unbalanced data, we changed the cost factor to balance positive and negative examples. All features will be first standardized to reduce

the impact of different scales of features.

As the entity-centric document filtering problem is expected to make a binary decision, we used classification-oriented metrics including precision, recall, F1-measure and accuracy as our evaluation criteria. For BoostMapping, we empirically set the number of feature mapping to be 150 and the predicate number to be 3 (later we would investigate their impact).

2.5.2 Quantitative Evaluation

Overall Performance

We introduce four baselines to incrementally validate the performance of the proposed LinearMapping and BoostMapping models. First, we experiment QueryByName to verify the necessity of leveraging information in the identification pages. Second, by comparing our models with QBD-TFIDF, we verify the necessity of automatically learning a keyword importance function. Third, we compare our models with VectorSim to demonstrate that our models better characterize the entity-relevance semantics than empirical similarity functions. Finally, we show that our models outperforms RelEntity, the best algorithm among all TREC-KBA submissions. The design details of baselines are introduced as follows,

1. *Query by Entity Name* (QueryByName). This baseline treats entity-centric filtering as a standard keyword-based information retrieval task by using only entity names as keyword queries. Following the standard feature design in the LETOR benchmark [60], we extract ranking features such as TFIDF and BM25 for each document (note that not all LETOR features are covered, as some features like PageRank are only available in commercial search engines), and train a ranking model by SVMRank [35], which is the state-of-the-art learning to rank technique, to predict document relevance. We then feed the predicted score as one feature into SVMLight, which will learn a threshold from training data and output binary results.

2. *Query by Document based on TFIDF Scoring* (QBD-TFIDF) [76]. This work studies the retrieval task when the query is a document. As the solution, QBD-TFIDF first extracts noun phrases from documents, and scores those phrases by a TF-IDF-based formula. The noun phrases (*e.g.*, top 20) with highest score will be chosen as an extended query. In [76], such queries are submitted to a commercial search engine to retrieve documents, while in our implementation, as our candidate documents are given, we follow the same procedure in QueryByName to extract features and learn a document ranker for relevance prediction.

3. *Keyword-Vector-based Similarity* (VectorSim). This baseline first represents each document by a TF-IDF keyword vector, and then calculates document similarity using five different metrics introduced in [31], including Euclidean distance, cosine similarity, Jaccard coefficient, Pearson correlation coefficient and averaged Kullback-Leibler Divergence. These measures will be fed into SVMLight as features to learn a

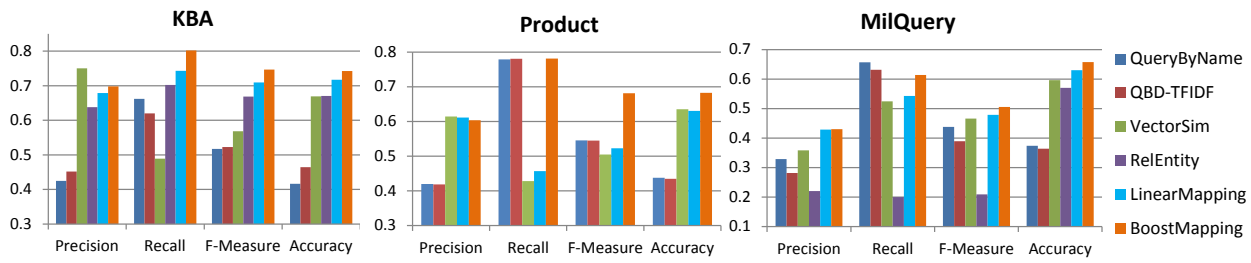


Figure 2.6: Performance comparison with baselines.

document filter.

4. *Related-Entity-based Scoring* (RelEntity) [49]. This is the algorithm that achieves the best performance among all TREC-KBA submissions. As the key idea, given an entity e, RelEntity first identifies its related entities $\hat{\mathbf{e}}$ which are also Wikipedia entities and appear in e's Wikipedia page, and then score document d by an emperical formula $S(d) = 100 * N(d, e) + N(d, \hat{\mathbf{e}})$, where $N(d, e)$ is the number of e in d. A threshold ($\mathcal{T} = 100$) is manually set to produce binary scores.

The result in Figure 2.6 demonstrates that BoostMapping significantly outperforms all four baselines. As an ideal model should achieve both high precision and high recall, we compare the model performance on F-measure and accuracy. In particular, BoostMapping achieves encouraging improvement against runner-up RelEntity in TREC-KBA (F-measure +11.7%, accuracy +10.7%), and runner-up VectorSim in Product (F-measure +35.0%, accuracy +7.4%) and MilQuery (F-measure +8.6% and accuracy +10.3%) with p-value< 0.05. LinearMapping also achieves improvements against runner-ups in the KBA dataset (F-measure +6.2%, accuracy +7.0%) and the MilQuery dataset (F-measure +2.9%, accuracy +5.7%), while obtains similar performance with the runner-up in the product dataset.

We analyze the performance difference between our models and other baselines in more details:

First, QueryByName does not perform well. We can observe that QueryByName tends to achieve high recall but very low precision in all three datasets. That's because traditional information retrieval techniques tend to regard a document as relevant if it mentions the query entity a lot, which might not be true if the document does not contain related information of the query entity. The comparison demonstrates that entity description pages do provide useful information to capture the sense of entity-relevance. Moreover, for ambiguous entities (*e.g.*, in TREC-KBA, there exist ambiguous entities such as "basic element [company]" and "basic element [music group]."), without leveraging their description pages, it is difficult for QueryByName to justify their relevant documents based on only entity names.

Second, although QBD-TFIDF leverages the information from the entity identification page by selecting keywords with high scores to extend the entity name query, its performance is very similar to QueryByName. As the scoring function of QBD-TFIDF is manually crafted, it might not well characterize the keyword

importance for an entity. The comparison between QBD-TFIDF and our models confirms the necessity of learning the keyword weighting function from the corpus.

Third, by comparing VectorSim with BoostMapping in Figure 2.6, we observe that, different from QueryByName and QBD-TFIDF, VectorSim tends to have high precision and low recall, in both KBA and product datasets. That is because assessors tend to label documents that are very similar to the entity identification page as relevant, and VectorSim could correctly identify those documents with high accuracy. However, there also exist many documents that are not similar to the entity identification page, and the failure in identifying these low-similarity documents leads to low recall for VectorSim.

Fourth, RelEntity achieves satisfactory performance in TREC-KBA, demonstrating that the number of related entities is a critically useful feature in TREC-KBA. However, such a method is not generalizable – in Product, the result of RelEntity is not listed, because it is unclear how to similarly define relevant entities in the context of Amazon product pages; in Mil-Query, RelEntity achieves poor performance, revealing that such a feature is no longer useful when the dataset is changed. The result shows it is difficult to adapt to different datasets by relying on only one feature; while in our framework, we can capture the same intuition of RelEntity by adding one meta-feature *AnchorEntity* as explained in Figure 2.2, and let the learning algorithm determine the importance of the meta-feature for different datasets.

In general, both LinearMapping and BoostMapping achieve satisfactory performance in three datasets, demonstrating the effectiveness of feature mapping for entity-centric document filtering. In particular the results on the Product and MilQuery dataset confirm our confidence that our models can not only handle different types of entity identification page, but also work well even if the query entities cover very different types.

LinearMapping versus BoostMapping

Compared with LinearMapping, with stronger representation power, the improvement of BoostMapping against other baselines is more significant. Since LinearMapping assumes that $weight(w, e)$ takes the linear form of meta-features, the performance of LinearMapping largely depends on whether the datasets follow such a linear assumption, while BoostMapping does not rely on such an assumption. To verify such a claim, for each meta-feature, we train a LinearMapping model, and use the prediction accuracy to estimate the linearity of the meta-feature, *i.e.*, how it fits the linear assumption. We then divide meta-features into two sets of the same size, containing meat-features with low linearity and high linearity respectively. The results of LinearMapping and BoostMapping over these two datasets (*e.g.*, denoted as LinearMapping-Low and LinearMapping-High) and all meta-features (*e.g.*, denoted as LinearMapping-All) are compared in Figure

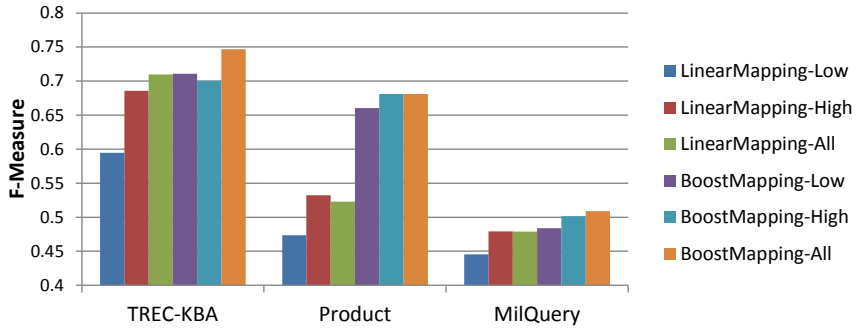


Figure 2.7: LinearMapping versus BoostMapping.

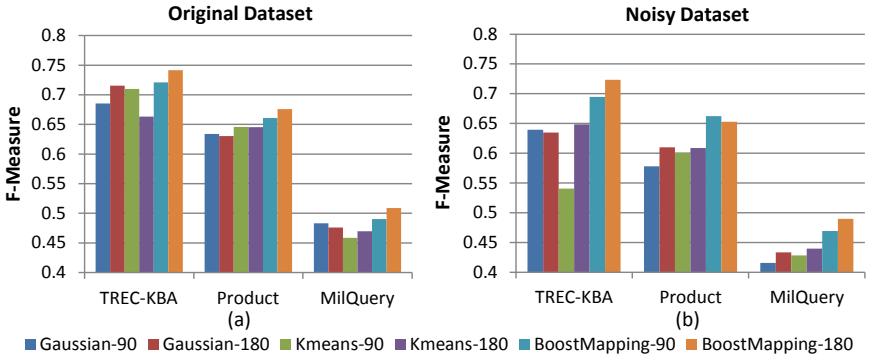


Figure 2.8: Other clustering algorithms.

2.7.

Figure 2.7 offers an insightful understanding of the differences between LinearMapping and BoostMapping. First, by comparing BoostMapping-Low with LinearWeight-Low, BoostMapping-Low achieves significant improvement in F-Measure (TREC-KBA +19.5%, Product +39.4%, MilQuery +8.6%), which verifies our claim in Section 2.4.3 that BoostMapping has stronger representation power compared with LinearMapping when the linear assumption does not hold. Second, LinearMapping-High achieves comparable performance with BoostMapping-High in both TREC-KBA and MilQuery, verifying our claim that LinearMapping could also achieve stratificatory performance over meta-features with high linearity; while for Product, even LinearMapping-High achieves low F-Measure, which implies that all meta-features in the product dataset have relatively low linearity and explains the general poor performance of LinearWeight-All on all meta-features. As BoostMapping could benefit from meta-features of different linearity, BoostMapping-All performs better compared with LinearMapping-All in F-measure (TREC-KBA +5.19%, Product +30.3%, MilQuery +6.22%).

Clustering Algorithms

As one important advantage of BoostMapping, it realizes distantly supervised clustering to guarantee the predictive power of the generated keyword clusters for entity-centric document filtering. To verify such

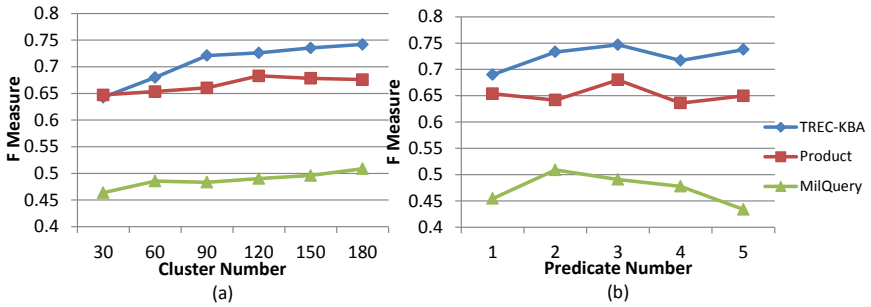


Figure 2.9: Different model parameters.

an advantage, we design two baselines based on with two standard clustering algorithms, K-Means and Gaussian Mixture model, which directly cluster keywords based on their meta-features without guidance from document labels, and then use SVMLight to learn the importance of each cluster. The performance of BoostMapping and two baselines with 90 and 180 clusters is compared in Figure 2.8(a). Furthermore, in order to compare these algorithms' performance when there exist some irrelevant meta-features, we build up a synthetic "noisy" dataset by adding 30 randomly generated meta-features for each keyword to the original dataset. The result on the synthetic noisy dataset is displayed in Figure 2.8(b).

As Figure 2.8(a) displays, when using original meta-features, BoostMapping outperforms K-Means and Gaussian with different cluster number in different dataset. For example, when the cluster number is 180, the improvement against the runner-up is 3.7% in TREC-KBA, 4.7% in Product and 5.3% in MilQuery. The result demonstrates that our designed meta-features are relevant with the task such that standard clustering algorithms following the same keyword mapping idea could achieve satisfactory performance. We are also interested in the robustness of BoostMapping when there exist some irrelevant meta-features. From Figure 2.8(b), we observe that the performance of KMeans-Noise and Gaussian-Noise degrade significantly due to the existence of noisy meta-features, while for BoostMapping-Noise, it is only slightly affected, which achieves large improvement compared with KMeans-Noise and Gaussian-Noise (11.6% in TREC-KBA, 8.6% in Product and 11.4% in MilQuery when the cluster number is 180). Such a comparison confirms our confidence about the robustness of BoostMapping – by using document labels as guidance, BoostMapping generates clusters that are more useful for the entity-centric filtering problem.

Different Parameters

In this section, we investigate the effect of cluster number M and predicate number L on the performance of BoostMapping. As Figure 2.9(a) shows, with respect to different number of clusters, we observe that BoostMapping's F-Measure increases a lot from 30 to 120 clusters in all three datasets (TREC-KBA +17.4%, Product +5.52%, MilQuery +5.73%), while from 120 to 180 clusters, the performance becomes much stabler.

Data	Entity	Important Keywords	Unimportant Keywords
TREC-KBA	Jim Steyer	Stanford advocacy clinic attorney	category 6 assist birth year
	Douglas Carswell	clacton sceptical anglia incumbent	encyclopedia kitchen blog
Product	Apple iPhone 4	similar nice easier ios5 slow 399	ipod ipad will now device
	Canon S95	stablizer hybrid capture zoom spec	music olympus believe 149
Mil-Query	hostage rescue	Tatics hostile explosion nuclear	www distraction verify dot
	Kodak	Photography camera sell transition	Tablet smartpone amazon

Figure 2.10: Keywords of different importance.

Figure 2.9(b) compares the performance of BoostMapping with different number of predicates. The result shows that BoostMapping usually achieves satisfactory performance when the predicate number is 2 or 3. Setting the predicate number to be 1 is insufficient, as it fails to capture the interaction between different meta-features, *e.g.*, keywords that appear in the infobox with high IDF values; while when the predicate number is large, each cluster might contain too few keywords and the model tends to overfit to some particular queries.

2.5.3 Case Study

To given an intuitive illustration of how meta-features help determine important keywords for entities, in Figure 2.10, we perform case studies by showing 2 example entities in each dataset, and listing some of their important/unimportant keywords with high/low $weight(w, e)$ returned by BoostMapping.

First, with respect to important keywords, we observe that, for TREC-KBA and MilQuery, BoostMapping usually returns keywords that characterize the basic information of the entity, for example, keywords "advocacy" and "attorney" for entity "Jim Steyer," as Jim Steyer is best known to be a child advocate and civil rights attorney. While for the Product dataset, BoostMapping returns not only keywords describing the product specification, *e.g.*, "stabilizer" and "hybrid" for "Canon S95", but also evaluation keywords such as "nice" and "easier" (mentioned in the review sections of Amazon product pages), because labelers tend to label documents including helpful comments of the products as relevant.

Second, for unimportant keywords, BoostMapping usually returns those keywords appearing in every Wikipedia page (*e.g.*, "category" and "encyclopedia"), or in some less important positions (*e.g.*, "blog" and "kitchen" in the reference section of Jim Steyer's Wikipedia page). For product entities, BoostMapping can successfully identify those keywords referring to closely related but different entities (*e.g.*, "ipod", "ipad" for entity "iphone", and "olympus" for "canon") as unimportant keywords, which are very helpful for recognizing those irrelevant documents, which occasionally mention about the target entity, and mainly discuss about other entities.

Chapter 3

Query By Entities: Cross-Task Document Scoring

3.1 Introduction

With much information in the world represented by unstructured text, many applications study how to develop document *scorers*, which can analyze the content of a document and determine its relevance for some information need, *e.g.*, text categorization [33], document retrieval [15]. Since manually crafting scorers is difficult, people are interested in how to automatically *learn* such scorers from labeled documents.

Scorer learning techniques have been actively studied for years – unlike early applications which usually deal with only one topic (*e.g.*, judging if a document is about "finance" [33]), with the prevalence of the Web serving the whole world of users and connecting numerous sources of data, most applications nowadays must handle various user needs, *i.e.*, *queries*, and diverse sources, *i.e.*, *domains*.

First, in terms of *queries*, many applications aim at ranking documents for queries that represent users' information needs, *i.e.*, realizing *learning to rank*. Unlike traditional IR which deals with short queries, learning to rank applications nowadays have to handle more sophisticated queries:
Application 1: *Verbose-Query-based Retrieval* [44, 7], which addresses verbose queries that consist of one or more long sentences. It is more challenging than traditional IR, because long queries usually contain extraneous terms which might hinder retrieval of relevant documents.
Application 2: *Entity-Centric Document Filtering* [80], which studies, given an entity (*e.g.*, a person) characterized by an identification page (*e.g.*, her Wikipedia page) as a query, how to identify documents relevant to the entity. Since an identification page is usually long and noisy, the solution has to identify keywords that represent the characteristics of the entity (*e.g.*, "microsoft" for entity "Bill Gates") for better retrieval accuracy.

Second, in terms of *domains*, traditional scorer learning techniques work well only when the training and testing documents use similar distributions of keywords, *i.e.*, documents are from the same domain. Due to the expensive cost of labeling, we expect to learn a document scorer that could be adapted across different domains, *i.e.*, realizing *domain adaptation*, for example:

Application 3: *Cross-Domain Sentiment Analysis* [10], which studies how to adapt a sentiment scorer across different domains of reviews. It is challenging because people tend to use different sentiment keywords in different domains, *e.g.*, "boring" in book reviews, and "leaking" in kitchen appliance reviews instead.

We observe that, although learning to rank and domain adaptation seem distinct from each other, both of them have to handle the varying importance of each *keyword* in different queries/domains (in this chapter, we do not consider other properties which are orthogonal to the keyword content, *e.g.*, pagerank, although they can be easily incorporated as well). For example, in entity-centric document filtering, keyword "Microsoft" is important for query "Bill Gates," but not for "Michael Jordan;" in sentiment analysis, as mentioned earlier, different domains would use different keywords to represent the same sentiment. Therefore, as the common challenge for these two types of applications, both of them have to consider how to bridge keywords across different queries or domains.

Problem – Cross-Task Document Scoring. As the *first contribution* of this chapter, observing such a common challenge, we propose to study a general *cross-task document scoring* problem, which, as far as we know, is the first attempt to unify learning to rank and domain adaptation. Formally, cross-task document scoring aims at learning a scorer $\mathcal{F}(d, t)$ to predict the relevance of document d for task t, where the notion of "*task*" represents the scoring of documents for one query in one domain. In the training phase, we are given some documents labeled for some tasks t (*e.g.*, some queries) to learn $\mathcal{F}(d, t)$; in the testing phase, as highlighted by "cross-task," $\mathcal{F}(d, t)$ should be capable of handling new tasks t' (*e.g.*, new queries) which do not appear in the labeled data.

Challenge – Learning to Adapt Keyword Contribution. As our *second contribution*, we identify the core challenge of cross-task document scoring as learning to adapt keyword contribution across tasks. Formally, if we use $Contrib(w, d, t)$ to denote the contribution of keyword w for document d with respect to task t, the relevance of document d is essentially the accumulation of its keywords' contribution $Contrib(w, d, t)$. Therefore, the challenge of learning $\mathcal{F}(d, t)$ lies in how to determine keyword contribution $Contrib(w, d, t)$ for tasks that do not appear in the training data, *i.e.*, *learning to adapt keyword contribution.*

Learning to adapt keyword contribution is critical for enabling cross-task scoring; however, traditional learning to rank frameworks (*e.g.*, RankSVM [36]) simply circumvent the problem. To learn a scorer, such frameworks adopt an ensembling idea to combine a set of manually crafted sub-scorers $f_k(d, t)$ as features, *e.g.*, in document retrieval [15, 48], $f_k(d, t)$ could be BM25, language model. These learning to rank frameworks are feasible for traditional document retrieval, as there exist many readily studied scorers that could be used as features; however, for newly proposed applications, such scorers are seldom available, and we need laborious feature engineering in order to adopt these frameworks.

Insight – Keyword Scoring Principle. As our *third contribution*, towards learning to adapt keyword contribution, we propose our key insight:

Keyword Scoring Principle – the importance of keyword w for document d with respect to task t, i.e., $Contrib(w, d, t)$, depends on: 1. the importance of keyword w for task t; 2. the importance of keyword w for document d.

Although such a principle is not abstracted before, it is intuitive and implicitly followed in the previous design of manually crafted scorers. Take BM25 as example: in terms of the first aspect, BM25 assumes that keywords with high inverse document frequency (*i.e.*, high *IDF*) are more important for the query; in terms of the second aspect, BM25 assumes that keywords mentioned a lot in the document (*i.e.*, high *TF*) should have higher contribution.

Abstract – Feature Decoupling. As our *fourth contribution*, to fulfill our goal of learning $Contrib(w, d, t)$ based on the principle, we propose the idea of *feature decoupling*, which suggests "decoupling" the original scorer-as-feature design in traditional learning to rank frameworks into two types of more elementary features:

To determine "the importance of keyword w for task t," we design *meta-features*, denoted by $f_k^{(M)}(w, t)$, to represent task-related keyword properties. In learning to rank, some recent works [6, 44, 8, 80] adopt such an idea, *e.g.*, using meta-features such as keyword position to identify important keywords from queries. In domain adaptation, Blitzer *et al.* [10] propose a model to use keyword correlation to bridge keywords from different domains. We can use meta-features to realize the same insight, which will be discussed in details in Section 3.3.4.

To determine "the importance of keyword w for document d," we propose the concept of *intra-features*, denoted by $f_k^{(I)}(w, d)$, to characterize how keyword w occurs in document d. The motivation is that, besides simply counting keywords as most applications do, we can characterize the keyword occurrence more generally and systematically for higher prediction accuracy. For example, keywords that appear in the title, anchor text or URL usually have larger contribution to the relevance.

Given such a decoupled feature design, we have to appropriately "re-couple" $f_k^{(M)}(w, t)$ and $f_k^{(I)}(w, d)$ to determine $Contrib(w, d, t)$, and $\mathcal{F}(d, t)$ finally takes the following abstraction:

$$\mathcal{F}(d, t) : \langle f_1^{(M)}(w, t), ...; f_1^{(I)}(w, d), ... \rangle \rightarrow \mathbb{R} \tag{3.1}$$

To the best of our knowledge, such a learning framework, which aims at learning a scorer upon two types of elementary features, has not been studied before (previous works [6, 44, 8, 80] which have the concept of meta-features do not model intra-features).

Towards learning keyword contribution based on decoupled features, our framework has to fulfill two requirements:

Requirement 1: Inferred Sparsity. Unlike traditional learning to rank frameworks [15, 48] which do not model the concept of keywords, our scorer should be aware of the potential interference from noisy keywords. For example, both verbose-query-based retrieval and entity-centric document filtering focus on long queries, in which many keywords are unrelated to user intent or target entities; in sentiment analysis, a review usually contains many keywords that are irrelevant to sentiment. Even if we assign such keywords with small contribution, their values, once accumulated, will still severely affect the document score.

Therefore, in order to filter noisy keywords, we require that the keyword contribution $Contrib(w, d, t)$ be *sparse*, which only outputs non-zero values for important keywords. Different from traditional sparse learning [72] which aims at learning sparse feature weightings to enforce feature sparsity, our goal is to *sparsify the inferred value of a function*, *i.e.*, achieving *inferred sparsity* (their difference will be further discussed in Section 3.4.1).

Requirement 2: Distant Supervision. Toward realizing learning to score keywords, another challenge arises from the lack of the keyword labels. In most applications, we are only provided with document labels, and it is impractical to request manual keyword labels for learning $Contrib(w, d, t)$. Therefore, we require that the scorer should fulfill *distant supervision*, by only using document labels to "distantly" guide the adaptation of keyword contribution.

Solution – Tree-Structured Restricted Boltzmann Machine. As our *fifth contribution*, to fulfill these two requirements, we propose a novel *Tree-structured Restricted Boltzmann Machine* (T-RBM) model for the cross-task document scoring problem.

First, to achieve *inferred sparsity*, the model needs a scoring scheme which can eliminate the contribution of noisy keywords. We develop a *two-stage* procedure: in the first stage, we learn a classifier to discretize the importance of keywords into different levels based on their meta-features, and *regularize* the classifier to enforce the elimination of noisy keywords; in the second stage, we determine the contribution of important keywords by their intra-features and set the contribution of unimportant ones to be zero.

Second, to achieve *distant supervision*, we propose to *join* the two stages in one model. Specifically, we take advantage of Markov Network to connect keyword importance and document relevance, such that we can use document labels to directly supervise the learning of the keyword classifier.

Based on these ideas, we design T-RBM, which, as a variant of Restricted Boltzmann Machine [70] (one type of bipartite Markov Network), models each document as a tree graph with the root node representing a document and the leaf nodes representing its keywords. Free of loop structures, T-RBM could be efficiently

trained by exact belief propagation [58].

In the experiments, we performed our evaluation on verbose-query-based retrieval, entity-centric document filtering and cross-domain sentiment analysis in three different datasets. By comparing T-RBM with four state-of-the-art baselines, we observed that our framework not only provides a conceptually unified modeling but also significantly improves the results on different applications.

3.2 Related Work

In terms of *abstraction*, learning to rank and domain adaptation are separately abstracted and studied in previous works, *e.g.*, document retrieval [15, 6, 44, 8], entity-centric document filtering [80], cross-domain sentiment analysis [10]. Inspired by their works, we identify their common challenge and propose a novel framework to unify these two problems for achieving a more general solution.

In terms of *challenge*, cross-task document scoring boils down to learning to adapt keyword contribution across different tasks.

1. For *learning to rank*, most previous works [15, 48] simply circumvent the challenge, leaving the burden of determining keyword contribution to feature designers. Some of the recent works [6, 44, 8, 80] start to confront the challenge by modeling keyword-level features, *i.e.*, meta-features, to learn keyword contribution; however, none of them explicitly model intra-features, failing to capture different kinds of keyword occurrences in the document.

2. For *domain adaptation*, Blitzer *et al.* [10] propose structural correspondence learning (SCL) to bridge keywords from different domains. The intuition is that, given two domain-specific keywords (*e.g.*, "boring" from book reviews and "leaking" from kitchen appliance reviews), if both of them co-occur a lot with some pivot keywords (*i.e.*, keywords like "bad," "worst" which are commonly used in all domains), these two keywords should share similar sentiment (*e.g.*, both "boring" and "leaking" represent negative sentiment). To realize such an insight, SCL learns a set of pivot predictors, each of which predicts the occurrence of one pivot keyword based on other domains-specific keywords, to relate different domains. Li *et al.* [46] and Pan *et al.* [56] follow the same intuition but use different approaches such as feature alignment [56] and matrix decomposition [46]. Different from these approaches which "hardcode" the logic of keyword adaptation in the model design, our feature decoupling idea allows designers to conveniently incorporate different ways of keyword adaptation by meta-features.

In terms of *technique*, we propose T-RBM, a novel two-stage Markov Network taking the decoupled features as input and fulfilling the requirements of inferred sparsity and distant supervision.

1. With respect to *inferred sparsity*, unlike sparse learning [72] which learns sparse parameters as feature weightings, we aim at sparsifying the inferred value of the keyword contribution function. Existing meta-feature-based solutions have different limitations in fulfilling this requirement. The solutions proposed by Lease *et al.* [44], Bendersky *et al.* [8] and Zhou *et al.* [80] do not fulfill the requirement, making the prediction result vulnerable to noisy keywords. Zhou *et al.* [80] propose another BoostMapping model, which achieves inferred sparsity by clustering keywords based on their meta-features and eliminating noisy clusters; as the limitations, BoostMapping is difficult to solve when we have to model multiple intra-features, and might overfit the training data. We will compare T-RBM with these models in details in Section 3.4.1.

2. With respect to the requirement of *distant supervision*, similar to T-RBM, Bendersky *et al.* [6] propose to learn a keyword classifier to discover important concepts for retrieval; however, their solution relies on the existence of keyword labels, which is impractical for most applications. Different from their work, taking the advantage of Markov Network, T-RBM manages to train the keyword classifier based on only document labels.

3. With respect to the *model structure*, the most related work to ours is the Markov Random Field model proposed by Lease *et al.* [44]. As the key difference, their solution is a generative model characterizing $P(q, d)$ – the joint probability of observing query q and document d, while our T-RBM model directly models the conditional probability $P(d|q)$. It has been repeatedly confirmed that a discriminative model usually yields better generalization performance compared with a generative model [73]; furthermore, as mentioned earlier, their solution does not fulfill the inferred sparsity requirement.

4. Restricted Boltzmann Machine (RBM), as one type of bipartite Markov Network, is adopted in many applications such as topic modeling [67], deep learning [29], *etc.*With distinct settings and objectives, our proposed T-RBM is different from traditional RBM models in two aspects: first, T-RBM takes a tree structure which can be trained efficiently; second, T-RBM adopts a novel hidden variable regularization technique, which allows the model to control the inferred sparsity of keyword contribution.

3.3 Cross-Task Document Scoring

In this section, we formally define the *cross-task document scoring* problem. To tackle the challenge of *learning to adapt keyword contribution*, we propose the insight of *keyword scoring principle* and the concept of *feature decoupling*.

3.3.1 Problem: Cross-Task Document Scoring

In document scoring, we aim at predicting the relevance of a document d for a particular task t. Formally, task t could be represented as a function $t : d \rightarrow \mathbb{R}$, which takes a document d as input, and outputs a score denoting the relevance of d. We define that two tasks t_1 and t_2 are different, if there exists one document d satisfying $t_1(d) \neq t_2(d)$. Therefore, identifying relevant documents for different queries belongs to two different tasks, as one document usually has different relevance for two queries; similarly, the sentiment judgement of a book review is also different from that of a kitchen appliance review.

In contrast with *single-task document scoring* which tackles only one task (*e.g.*, text categorization [33] learns a scorer to predict if a document is about a fixed topic such as "finance"), in a *cross-task document scoring* problem, we are interested in a set of different but related tasks $\mathcal{T}$. For example, in verbose-query-based retrieval and entity-centric document filtering, each $t \in \mathcal{T}$ represents a task of predicting document relevance for one verbose query or one entity; in cross-domain sentiment analysis, each task t is to judge the sentiment of reviews form one particular domain.

Formally, in *cross-task document scoring*, our goal is to automatically learn a document scorer $\mathcal{F}(d, t)$, which could output the relevance of document d for task $t \in \mathcal{T}$.

In *the training phrase*, we are given a set of training documents $\mathbf{d} = \{d_1, d_2, ..., d_N\}$ and a list of document labels $\hat{\mathbf{y}} = \{\hat{y}_1, ..., \hat{y}_N\}$, where each d_i is labeled by $\hat{y}_i$ denoting the relevance of d_i for task $t_i \in \mathcal{T}$ (t_i and t_j could refer to the same task, indicating d_i and d_j are labeled for the same query or in the same domain). Here, $\hat{y}_i$ could be either binary (*e.g.*, 0–negative, 1–positive) or ordinal (*e.g.*, 0–irrelevant, 1–relevant or 2–perfectly relevant).

Based on the training documents, we aim at learning $\mathcal{F}$, which, in *the testing phase*, could predict the relevance of a new document d' for task $t' \in \mathcal{T}$. Specifically, as highlighted by "cross-task," we require that t' should differ from all the training tasks, *i.e.*, $t' \neq t_i$ for all i. That is because, in most applications, the target task set $\mathcal{T}$ could not be covered by finite training examples (*e.g.*, there are infinite possible queries in document retrieval) and thus, $\mathcal{F}$ should be adaptable to unseen queries or domains, *i.e.*, new tasks.

3.3.2 Challenge: Learning to Adapt Keyword Contribution

To score a document d for a task t, the scorer has to assess the content of document d. Formally, we use $\mathcal{V}(t)$ to represent the vocabulary of keywords that are considered in task t, where each keyword $w \in \mathcal{V}(t)$ could be 1-gram, 2-gram, noun phrases, *etc.* Following previous works [8, 80, 10], in verbose-query-based retrieval and entity-centric document filtering, $\mathcal{V}(t)$ covers keywords that are mentioned in the query or the entity identification page, while in cross-domain sentiment analysis, $\mathcal{V}(t)$ includes all the keywords. We then

define $\mathcal{W}(d,t) \subseteq \mathcal{V}(t)$ as the content of document d that is related with task t.

Based on the document content $\mathcal{W}(d,t)$, the relevance of document d could be viewed as the accumulation of its containing keywords' contribution. Formally, if we use $Conrib(w,d,t)$ to denote the contribution of keyword w to document d with respect to task t, $\mathcal{F}(d,t)$ takes the form of

$$\mathcal{F}(d,t) \propto \sum_{w \in \mathcal{W}(d,t)} Conrib(w,d,t) \tag{3.2}$$

As Eq. 3.2 shows, the challenge of cross-task document scoring becomes how to learn $Conrib(w,d,t)$ to determine the keyword contribution for new tasks t' which do not appear in the training data, *i.e.*, *learning to adapt keyword contribution*.

3.3.3 Insight: Keyword Scoring Principle

Due to the requirement of "cross-task," the learning of $Conrib(w,d,t)$ is non-trivial. If our target is a single-task document scoring problem (*i.e.*, tackling the same task in both training and testing phases), as a common solution [33], we can define $Conrib(w,d,t) = \alpha_w * TF_w(d)$, where each feature function $TF_w(d)$ represents how many times document d contains a specific keyword w, and α_w, as its weighting, characterizes the importance of keyword w (*e.g.*, in text categorization , if the target topic is about finance, keyword "stock" should have value of α_w). The scorer is then defined by

$$\mathcal{F}(d,t) = \sum_{w \in \mathcal{W}(d,t)} \alpha_w * TF_w(d) \tag{3.3}$$

where α_w could be learned by standard learners. However, such a design could not be applied in cross-task document scoring, because, as we mentioned in Section 3.1, one keyword usually has very different importance for different tasks, and thus, α_w learned from training data could not be adapted to new tasks.

Traditional learning to rank frameworks (*e.g.*, RankSVM [36]) manage to tackle different tasks (*i.e.*, queries); however, they circumvent the problem of learning $Conrib(w,d,t)$, which define the scorer by

$$\mathcal{F}(d,t) = \sum_{k=1}^{N} \beta_k f_k(d,t) \tag{3.4}$$

where $f_{k \in \{1 \ldots N\}}(d,t)$ is a set of handcrafted sub-scorer features and β_k is learned to represent the confidence of $f_k(d,t)$, *e.g.*, in document retrieval [15], $f_k(d,t)$ could be vector space model, BM25 and language model. As we discussed in Section 3.1, such frameworks require designers to manually determine the keyword contribution in the feature design, laying heavy burden on designers.

In order to automatically learn the contribution of keywords, as our key insight, we propose the keyword scoring principle:

Definition 1 (Keyword Scoring Principle): If we use $\mathcal{R}_t(w,t)$ to denote the importance of keyword w for task t, and $\mathcal{R}_d(w,d)$ to denote the importance of keyword w for document d, the keyword contribution $Contrib(w,d,t)$ should take the following form:

$$Contrib(w,d,t) : \mathcal{R}_t(w,t), \mathcal{R}_d(w,d) \rightarrow \mathbb{R} \tag{3.5}$$

3.3.4 Abstract: Feature Decoupling

As we introduced in Section 3.1, towards automatically learning the keyword contribution $Contrib(w,d,t)$ based on the principle, we propose the concept of *feature decoupling*, which suggest "decoupling" the original scorer-as-feature design (*i.e.*, $f_k(d,t)$ used in Eq. 3.4) into two types of more elementary features:

Definition 2 (Feature Decoupling): To learn $Contrib(w,d,t)$, we propose to design, first, *meta-features* $f_k^{(M)}(w,t)$, which characterize task-t-related properties of keyword w, for determining the importance of keyword w for task t, *i.e.*, defining $\mathcal{R}_t(w,t)$ by

$$\mathcal{R}_t(w,t) : \langle f_1^{(M)}(w,t), f_2^{(M)}(w,t), ... \rangle \rightarrow \mathbb{R} \tag{3.6}$$

and, second, *intra-features* $f_k^{(I)}(w,d)$, which describe how document d contains keyword w, for determining the importance of keyword w for document d, *i.e.*, defining $\mathcal{R}_d(w,d)$ by

$$\mathcal{R}_d(w,d) : \langle f_1^{(I)}(w,d), f_2^{(I)}(w,d), ... \rangle \rightarrow \mathbb{R}. \tag{3.7}$$

Figure 3.1 lists all the features we use for verbose-query-based retrieval, entity-centric document filtering, and cross-domain sentiment analysis.

First, for *meta-features*, the designs vary a lot across applications:

a) In verbose-query-based retrieval, meta-features are designed to identify important keywords in the query. For example, *QueryPos* is used, as keywords mentioned earlier in the query tend to be more important. Following previous works [8], we model not only unigram, but also adjacent bigrams in the query. To discriminate their importance, we design separate features for them, *e.g.*, *QueryPos*[unigram] and *QueryPos*[bigram].

Name	Description
Meta-Feature $f_k^{(M)}(w,t)$	
QueryTF	Term Frequency of w is in the query (or the Wiki page).
IDF	The inverse term frequency of w
IsNoun (Vb/ Adj/ Adv/Num)	If w is a noun/verb/adjective/adverb/number
QueryPos	The first position where w is mentioned.
TFInTitle (InfoBox/OpenPara /AnchorEntity)	Term Frequency of w in the Wikipedia title/ the InfoBox/ the Wikipedia opening paragraph/ the anchor entities.
Corr[w_p]	Pearson Correlation between w and one pivot keyword w_p
Intra-Feature $f_k^{(I)}(w,d)$	
DocTF(Raw/Normalized/Log -Scaled)	Term frequency of w in d, which is represented by three features: raw frequency, frequency normalized by length, and logarithmically scaled frequency computed by log (f(w,d) + 1).
AnchorTF(Raw/...)	Term frequency of w in the anchor text of d
TitleTF(Raw/...)	Term frequency of w in the title of d
WindowsTF(Raw/...)	Only for Bigram. The number of times two keywords appear within a fixed size of window.
Doc-Feature $f_k^{(D)}(d)$	
DocLen	The document length of d
PivotTF[w_p]	Term Frequency of pivot keyword w_p in document d

Type	Feature List
Verbose-Query-based Retrieval	
Meta-Feature	[UniGram/Bigram] Q*ueryTF, IDF, QueryPos* [UniGram] *IsNoun(Vb/...)*
Intra-Feature	[UniGram/Bigram] *DocTF(Raw/...), AnchorTF(Raw/...), TitleTF(Raw/...)* [Bigram] WindowTF (Raw/...)
Doc-Feature	*DocLen*
Entity-centric document Filtering	
Meta-Feature	*QueryTF, IDF, IsNoun(Vb/ ...), QueryPos, TFInTitle(InfoBox/...)*
Intra-Feature	*DocTF(Raw/...), DocAnchorTF(Raw/...), DocTitleTF(Raw/...)*
Doc-Feature	*DocLen*
Cross-domain Sentiment Analysis	
Meta-Feature	*Corr[w_p]*
Intra-Feature	*DocTF(Raw/...)*
Doc-Feature	*DocLen, PivotTF[w_p]*

Figure 3.1: Feature design for three applications.

b) In entity-centric document filtering, we only model unigrams, because the query is already very long, and we find that considering bigrams does not help improve the performance. In additional to the features used in verbose-query-based retrieval, we also design meta-features like $TFInTitle$, $TFInInfoBox$, to leverage the structure of Wikipedia pages for identifying importance keywords.

c) In cross-domain sentiment analysis, the design of meta-features is very different from the other applications. In order to realize the insight proposed by Blitzer *et al.* [10] (introduced in Section 3.2), given a domain-specific keyword w, we propose to calculate a list of meta-features $Corr[w_p]$, each of which measures the Pearson correlation between w and one particular pivot keyword w_p – the correlation is positive if two

Training Phase:
Given: Training documents $\boldsymbol{d} = \{d_1, d_2, \ldots, d_N\}$, each d_i is labeled by y_i denoting the relevance of d_i with respect to task t_i, represented by its content $\mathcal{W}(d_i, t_i)$, and characterized by meta-features $f_k^{(M)}(w, t_i)$, intra-features $f_k^{(I)}(w, d_i)$ and doc-features $f_k^{(D)}(d_i)$
Output: A document scorer $\mathcal{F}(d, t)$ learned from training data.
Testing Phase:
Given: A new task t' ($t' \neq t_i$, for all i), document d' represented by $\mathcal{W}(d', t')$, features $f_k^{(M)}(w, t')$, $f_k^{(I)}(w, d')$ and $f_k^{(D)}(d')$
Output: The relevance of d' with respect to t', i.e., $\mathcal{F}(d', t')$

Figure 3.2: Cross-task document scoring.

keywords are positively correlated, negative if negatively correlated and zero if independent.

Second, *intra-features* characterize how a document contains a keyword. As discussed in Section 3.1, our designed intra-features describe how a keyword appears in different positions of a document (*e.g.*, title, anchor text) by different representations of term frequency (*e.g.*, term frequency normalized by document length). In verbose-query-based retrieval, as suggested by Bendersky *et al.* [8], we also design *WindowTF* specifically for bigrams, which counts the number of times that two keywords appear within a fixed size of window.

Finally, we design doc-features $f_k^{(D)}(d)$, which characterize document features that have the same weightings across different tasks. For example, in cross-domain sentiment analysis, we design $PivotTF[w_p]$ for each pivot keyword w_p (*e.g.*, "good," "bad"), which represents the same sentiment in all domains. The design of doc-features is not the main focus of this chapter, as they are the same with the features used in traditional learning models.

Given such a decoupled feature design, the cross-task document scoring problem is summarized in Figure 3.2.

3.4 Tree-Structured Restricted Boltzmann Machine

Towards realizing learning to adapt keyword contribution, we require that our framework fulfill the requirements of *inferred sparsity* and *distant supervision*. As our solution, we propose *Tree-Structured Restricted Boltzmann Machine* (T-RBM) to learn a two-stage document scorer based on the decoupled features.

3.4.1 Requirement: Inferred Sparsity & Distant Supervision

As we motivated in Section 3.1, in many applications, the keyword vocabulary $\mathcal{V}(t)$ we model is very noisy. If such noisy keywords are not appropriately filtered, their inferred values of $Contrib(w, d, t)$, once accumulated, will severely affect the prediction accuracy of $\mathcal{F}(d, t)$. Therefore, we require that our framework achieve *inferred sparsity*:

Requirement 1 (Inferred Sparsity for Keyword Contribution): $Contrib(w, d, t)$ should equal to zero for unimportant keywords w. ∎

This concept of inferred sparsity is closely related with feature sparsity, which traditional sparse learning works [72] aim to achieve. The goal of feature sparsity is to learn a sparse feature weighting vector, in which only a few features have non-zero weightings, for the purpose of reducing model complexity and increasing prediction accuracy. One common technique for achieving feature sparsity is l_1 regularization. For example, in single-task document scoring (which defines $\mathcal{F}(d, t)$ by Eq. 3.3), we can add an l_1 regularization term $|\alpha_w|_1$ to the objective function to learn sparse feature weightings α_w.

In contrast with feature sparsity which aims at learning sparse feature weightings, in inferred sparsity, we would like to "sparsify" the inferred value of a function, *i.e.*, $Contrib(w, d, t)$. Traditional feature sparsity techniques simply do not work, because the value of $Contrib(w, d, t)$ is jointly determined by a set of decoupled features – setting some of their weightings to be zero can not make the whole function become sparse.

Besides inferred sparsity, due to the lack of keyword labels, the learning framework should also achieve *distant supervision*:

Requirement 2 (Distant Supervision by Document Labels): The learning of keyword contribution $Contrib(w, d, t)$ should be distantly guided by only document labels $\hat{\mathbf{y}}$. ∎

As we introduced in Section 3.2, some existing works [8, 44, 80] on verbose-query-based retrieval and entity-centric document filtering also adopt the concept of meta-features. Although we can extend their works to support multiple intra-features, their models still have different limitations in fulfilling our requirements.

First, the concept weighting model proposed by Bendersky *et al.* [8], the Markov random field model proposed by Lease *et al.* [44] and the linear weighting model proposed by Zhou *et al.* [80] all belong to same the category, which define $Contrib(w, d, t) = [\sum_i \beta_i * f_i^{(M)}(w, t)] * DocTF(w, d)$ as a linear function over meta-features. Since $DocTF(w, d)$ is just one type of intra-feature, we can easily extend it to support

multiple intra-features, by defining $Contrib(w,d,t) = \sum_{i,j} \beta_{ij} * f_i^{(M)}(w,t) * f_j^{(I)}(w,d)$, and the scorer becomes

$$\mathcal{F}(d,t) = \sum_{i,j} \beta_{ij} \sum_{w \in \mathcal{W}(d,t)} f_i^{(M)}(w,t) * f_j^{(I)}(w,d) \tag{3.8}$$

where each $\sum_{w \in \mathcal{W}(d,t)} f_i^{(M)}(w,t) * f_j^{(I)}(w,d)$ could be treated as a generated feature, and β_{ij} could be learned by standard learners.

Such a linear design of $Contrib(w,d,t)$ fails to fulfill the requirement of inferred sparsity. Actually, except the trivial solution which sets all β_{ij} to be 0, for other assignments of β_{ij}, we can only get a small number of keywords (no more than the number of β_{ij}) which have zero contribution. The drawback of failing to achieve inferred sparsity could be observed by checking the semantics of the generated features. Take meta-feature *QueryPos* and intra-feature *DocTF* as example. The generated feature denotes the *QueryPos* summation of keywords from a document, which fails to capture the intuition that keywords with low *QueryPos* are more important, and documents containing more low-*QueryPos* keywords are more relevant.

Second, realizing the limitations of linear models, Zhou *et al.* [80] propose BoostMapping, which first adopts a boosting framework to generate a set of clusters $c_1, c_2, ...$, with each cluster c_i containing keywords sharing similar meta-features, *e.g.*, "$IDF > 10$ and $QueryPos < 5$," and then assumes that keywords from the same cluster c_i share the same contribution $Contrib_i(w,d,t) = \gamma_i * DocTF(w,d)$. Similar to the linear weighting model, we can extend it to support multiple intra-features by defining $Contrib_i(w,d,t) = \sum_j \gamma_{ij} * f_j^{(I)}(w,d)$, and the scorer becomes

$$\mathcal{F}(d,t) = \sum_{i,j} \gamma_{ij} \sum_{w \in \mathcal{W}(d,t) \cap c_i} f_j^{(I)}(w,d). \tag{3.9}$$

BoostMapping manages to achieve inferred sparsity, as the learner will assign $\gamma_{ij} = 0$ for unimportant clusters. However, it is difficult to extend the learning algorithm of BoostMapping (specifically, for generating clusters c_i) to support multiple intra-features. Furthermore, as reported in the paper [80] and confirmed in our experiment, BoostMapping might overfit to training tasks in some specific settings, which would lead to poor generalization capability.

3.4.2 Proposal: Two-Stage Scoring Model

To fulfill these two requirements, we develop a *two-stage* scoring procedure. Specifically, to achieve inferred sparsity, we *discretize* the importance of a keyword into different levels, and explicitly define $Contrib(w,d,t) = 0$ for unimportant keywords; to realize distant supervision, we unify the two stages in one single model, and

learn the feature weightings by only document labels. The model is described in the following:

• In the first stage, we build a keyword classifier $\mathcal{C}(w,t) \in \{0,1,...,L\}$ based on meta-features $f_k^{(M)}(w,t)$ to discretize the importance of keywords, where $\mathcal{C}(w,t) = 0$ indicates that w is a noisy keyword for task t, and $\mathcal{C}(w,t) \in \{1,...,L\}$ represents keywords of different importance levels.

• In the second stage, we determine contribution of each keyword w based on its importance level $\mathcal{C}(w,t)$. For important keywords with $\mathcal{C}(w,t) = l \neq 0$, the value of $Contrib(w,d,t)$ should depend on intra-features $f_k^{(I)}(w,d)$, while for unimportant ones (*i.e.*, $\mathcal{C}(w,t) = 0$), we set $Contrib(w,d,t) = 0$ to fulfill the requirement of inferred sparsity.

Formally, such a two-stage model defines $Contrib(w,d,t)$ as,

$$Contrib(w,d,t) = \begin{cases} \mathcal{U}_l(w,d) & \text{if } \mathcal{C}(w,t) = l \neq 0; \\ 0 & \text{if } \mathcal{C}(w,t) = 0. \end{cases} \tag{3.10}$$

where $\mathcal{U}_l(w,d)$ is a function defined over intra-features $f_k^{(I)}(w,d)$.

3.4.3 Solution: Tree-Structured Restricted Boltzmann Machine

In order to realize the design of $Contrib(w,d,t)$ in Eq. 3.10, the model should, first, characterize how keyword classifier $\mathcal{C}(w,t)$ depends on meta-features $f^{(M)}(w,t)$, second, determine how the contribution of important keywords, *i.e.*, $\mathcal{U}(w,d)$, is defined based on intra-features $f^{(I)}(w,d)$, and, third, be capable of learning $\mathcal{C}(w,t)$ in the absence of keyword labels. To achieve these three goals, we propose a novel *Tree-structured Restricted Boltzmann Machine* (T-RBM), in the framework of Markov Network, as our solution. Restricted Boltzmann Machine (RBM) [70] refers to one type of bipartite Markov network, which is widely used in many applications (*e.g.*, deep learning [29], topic modeling [67]). As a simplified RBM, our proposed T-RBM takes a tree structure to characterize the dependency between documents and keywords.

We highlight three important features of our proposed T-RBM model. First, to tackle the lack of keyword labels, T-RBM models the importance of keywords as *hidden variables*, and only uses document labels to guide the learning. Second, T-RBM adopts a novel *hidden variable regularization* idea, which allows the model to control the inferred sparsity of keyword contribution. Third, taking a tree structure, T-RBM could be *efficiently learned* by standard optimization techniques.

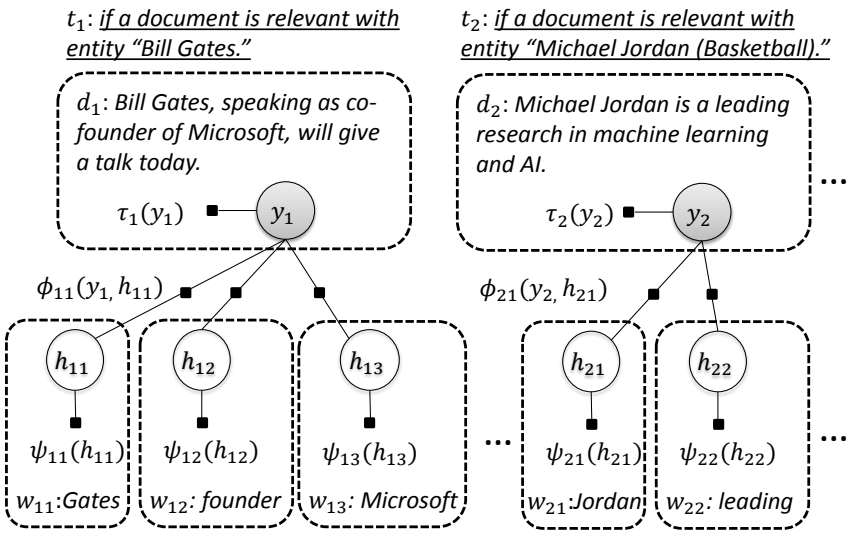


Figure 3.3: Tree-structured Restricted Boltzmann Machine.

Model Overview

Figure 3.3 shows a T-RBM model designed for entity-centric document filtering. Generally, T-RBM is composed of N tree-structured Markov Networks, each of which corresponds to one document. There are no edges between trees, indicating that the documents are independent with each other. Each tree contains two types of nodes y_i and $\mathbf{h_i} = \{h_{i1}, ..., h_{i|\mathbf{h_i}|}\}$, with $y_i \in \{0, 1\}$ denoting the relevance of document d_i ($y_i = 1$ if d_i is relevant, and $y_i = 0$ otherwise) and $h_{ij} \in \{0, 1, ..., L\}$ is a hidden variable denoting the importance of w_{ij} for task t_i (for notational convenience, we use $\mathbf{w_i} = \{w_{i1}, w_{i2}, ...\}$ to represent document content $\mathcal{W}(d_i, t_i)$, where w_{ij} denotes the j-th keyword in document d_i). More specifically, $h_{ij} = 0$ indicates that w_{ij} is unimportant, and $h_{ij} \in \{1, ..., L\}$ corresponds to different importance levels of keywords, *e.g.*, in sentiment analysis, both "good" and "bad" are important keywords, but have different contribution to the document sentiment.

Three types of factors $\psi_{ij}(h_{ij})$, $\phi_{ij}(y_i, h_{ij})$ and $\tau_i(y_i)$ are defined in T-RBM. Specifically, $\psi_{ij}(h_{ij})$ characterizes whether keyword w_{ij} is important for task t_i, $\phi_{ij}(y_i, h_{ij})$ models how keyword w_{ij} contributes its importance to document d_i, and $\tau_i(y_i)$ models the effect of other document features $f_k^{(D)}(d_i)$. Based on the factors, the conditional probability $P(y_i, \mathbf{h_i}|d_i, \mathbf{w_i}, t_i)$ is defined to represent the probability of a specific assignment of y_i and $\mathbf{h_i}$, given in the following,

$$P(y_i, \mathbf{h_i}|d_i, \mathbf{w_i}, t_i) \propto \tau_i(y_i) \prod_{j=1}^{|\mathbf{w_i}|} \phi_{ij}(y_i, h_{ij}) \psi_{ij}(h_{ij}) \tag{3.11}$$

We are interested in document scorer $\mathcal{F}(d_i, t_i)$, which, in the language of probability, is formally described by $P(y_i = 1|d_i, \mathbf{w_i}, t_i)$. Based on Eq. 3.11, we can represent $P(y_i = 1|d_i, \mathbf{w_i}, t_i)$ in the form of graph factors,

by marginalizing $P(y_i = 1, \mathbf{h_i}|d_i, \mathbf{w_i}, t_i)$ over all possible assignments of $\mathbf{h_i}$,

$$P(y_i = 1|d_i, \mathbf{w_i}, t_i) = \sum_{\mathbf{h_i}} P(y_i = 1, \mathbf{h_i}|d_i, \mathbf{w_i}, t_i) \tag{3.12}$$

To enforce inferred sparsity, we also want to control the percentage of noisy keywords in the model learning. In the language of probability, the inferred sparsity of keyword contribution could be represented by $P(\mathbf{h_i} = 0|d_i, \mathbf{w_i}, t_i)$, defined as follows,

$$P(\mathbf{h_i} = 0|d_i, \mathbf{w_i}, t_i) = \sum_{y=0}^{1} \prod_{j=1}^{|\mathbf{w_i}|} P(y_i = y, h_{ij} = 0|d_i, \mathbf{w_i}, t_i) \tag{3.13}$$

Factor Design

In T-RBM, we design factors $\psi_{ij}(h_{ij})$, $\phi_{ij}(y_i, h_{ij})$ and $\tau_i(y_i)$ to characterize the dependency required by Eq. 3.10.

First, $\psi_{ij}(h_{ij})$ characterizes how keyword classifier $\mathcal{C}(w_{ij}, t_i)$ judges if w_{ij} is an important keyword for task t_i, which should depend on meta-features $f_k^{(M)}(w_{ij}, t_i)$:

$$\psi_{ij}(h_{ij}) = \begin{cases} e^{\sum_k \theta_{kl}^{(M)} f_k^{(M)}(w_{ij}, t_i)} & \text{if } h_{ij} = l > 0; \\ 1 & \text{if } h_{ij} = 0. \end{cases} \tag{3.14}$$

Eq. 3.14 characterizes different levels of important keywords by defining L sets of feature weightings $\theta_{k,l}^{(M)}$. Note that we set $\tau_i(0) = 1$ since only relative value between $\tau_i(l)$ and $\tau_i(0)$ matters in the Markov Network.

Second, $\phi_{ij}(y_i, h_{ij})$ models how noisy keywords are filtered, and how the contribution of important keywords $\mathcal{U}_l(w, d)$ depends on intra-features $f_k^{(I)}(w_{ij}, d_i)$,

$$\begin{aligned} &\phi_{ij}(y_i, h_{ij}) \\ = &\begin{cases} \exp[\sum_k \theta_{kl0}^{(I)} f_k^{(I)}(w_{ij}, d_i)] & \text{if } h_{ij} = l \ \&\ y_i = 0; \\ \exp[\sum_k \theta_{kl1}^{(I)} f_k^{(I)}(w_{ij}, d_i)] & \text{if } h_{ij} = l \ \&\ y_i = 1; \\ 1 & \text{if } h_{ij} = 0. \end{cases} \end{aligned} \tag{3.15}$$

We set $\phi_{ij}(0, 0) = \phi_{ij}(1, 0) = 1$ to represent that if w_{ij} is an unimportant keyword, w_{ij} would not contribute any score to document d_i. When w_{ij} is important, *i.e.*, $h_{ij} = l > 0$, the value of $\phi_{ij}(y_i, h_{ij})$ depends on how w_{ij} appears in document d_i, *i.e.*, intra-features $f_k^{(I)}(w_{ij}, d_i)$. It should be noted that, even if we assume only one keyword importance level with $L = 1$, such a factor design can still discriminate keywords of different

importance, because the contribution of keyword w_{ij} is the marginalization result of $P(y_i, h_{ij} = 1|d_i, \mathbf{w_i}, t_i)$ and $P(y_i, h_{ij} = 0|d_i, \mathbf{w_i}, t_i)$.

Finally, we define $\tau_i(d_i)$ to incorporate document features $f_k^{(D)}(d_i)$ that are generalizable across tasks:

$$\tau_i(y_i) = \begin{cases} \exp[\sum_k \theta_k^{(D)} f_k^{(D)}(d_i)] & \text{if } y_i = 1; \\ 1 & \text{if } y_i = 0. \end{cases} \tag{3.16}$$

We use $\boldsymbol{\theta} = \{\theta^{(M)}_{1..K^{(M)},1..L}, \theta^{(I)}_{1..K^{(I)},1..L,0..1}, \theta^{(D)}_{1..K^{(D)}}\}$ to denote the set of parameters that need to be determined in T-RBM, where $K^{(M)}$, $K^{(I)}$ and $K^{(D)}$ denote the number of designed meta-features, intra-features and doc-features respectively, and the total number of parameters is $K^{(M)} * L + K^{(I)} * L * 2 + K^{(D)}$.

Model Learning

Following the maximal likelihood principle, our objective is to learn $\boldsymbol{\theta}$ to maximize the likelihood $\mathcal{L}(\boldsymbol{\theta}; \hat{\mathbf{y}})$ of observing document labels $\hat{\mathbf{y}}$, regularized by $\mathcal{G}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P(\mathbf{h_i} = \mathbf{0}|d_i, \mathbf{w_i}, t_i)$ to control the inferred sparsity of all the keywords. Formally the likelihood is defined by

$$\begin{aligned} &\operatorname*{argmax}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \hat{\mathbf{y}}) + \lambda \mathcal{G}(\boldsymbol{\theta}) \\ =& \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log P(y_i = \hat{y}_i|d_i, \mathbf{w_i}, t_i) + \lambda \log P(\mathbf{h_i} = \mathbf{0}|d_i, \mathbf{w_i}, t_i) \\ =& \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log \sum_{\mathbf{h_i}} \tau_i(\hat{y}_i) \prod_{j=1}^{|\mathbf{w_i}|} \phi_{ij}(\hat{y}_i, h_{ij}) \psi_{ij}(h_{ij}) \\ &+\lambda \log \sum_{y=0}^{1} \tau_i(y) \prod_{j=1}^{|\mathbf{w_i}|} \phi_{ij}(y, 0) \psi_{ij}(0) + (1 + \lambda) \log \sum_{y=0}^{1} \sum_{\mathbf{h_i}} \tau_i(y) \\ &\cdot \prod_{j=1}^{|\mathbf{w_i}|} \phi_{ij}(y, h_{ij}) \psi_{ij}(h_{ij}) \end{aligned} \tag{3.17}$$

where λ controls the inferred sparsity of keyword contribution. Specifically, when $\lambda > 0$, the model will favor more sparse keyword contribution, and vice versa. Here we first study document scoring as a classification problem by assuming that document labels are all binary (*i.e.*, $\hat{y}_i \in \{0, 1\}$), and will later extend our solution to document ranking problems which accept ordinal labels.

To optimize the objective function, we use the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm, which is a popular gradient-based method for solving unconstrained nonlinear problems – specifically, we adopt the LibLBFGS library [54], a c-implementation of L-BFGS. As the optimization

routine, in each step, we compute the current gradient value of the objective function at $\boldsymbol{\theta}$, and LibLBFGS will update $\boldsymbol{\theta}$ according to the gradient value. The optimization routine stops until the objective function converges.

The gradient value of the objective function at $\boldsymbol{\theta}$, as required by LibLBFGS, is computed as follows,

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\theta}; \hat{\mathbf{y}})}{\partial \theta_{kl}^{(M)}} =& \sum_{i=1}^{N} \sum_{j=1}^{|\mathbf{w_i}|} [P(h_{ij} = l | y_i = \hat{y}_i, d_i, \mathbf{w_i}, t_i) \\ & -(1+\lambda) P(h_{ij} = l | d_i, \mathbf{w_i}, t_i)] f_k^{(M)}(w_{ij}, t_i)\end{aligned} \tag{3.18}$$

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\theta}; \hat{\mathbf{y}})}{\partial \theta_{kly}^{(I)}} =& \sum_{i=1}^{N} \sum_{j=1}^{|\mathbf{w_i}|} [P(h_{ij} = l, y_i = y | y_i = \hat{y}_i, d_i, \mathbf{w_i}, t_i) \\ & -(1+\lambda) P(h_{ij} = l, y_i = y | d_i, \mathbf{w_i}, t_i)] f_k^{(I)}(w_{ij}, d_i)\end{aligned} \tag{3.19}$$

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\theta}; \hat{\mathbf{y}})}{\partial \theta_{k}^{(D)}} =& \sum_{i=1}^{N} [P(y_i = 1 | y_i = \hat{y}_i, d_i, \mathbf{w_i}, t_i) \\ & -(1+\lambda) P(y_i = 1 | d_i, \mathbf{w_i}, t_i)] f_k^{(D)}(d_i)\end{aligned} \tag{3.20}$$

In Eq. 3.18, Eq. 3.19 and Eq. 3.20, all the probabilities could be computed by belief propagation [58]. As T-RBM is a tree-structured graph, belief propagation could efficiently compute the exact result.

In general, the time complexity of T-RBM is $O[T * (N_d * K^{(D)} + N_e * K^{(I)} + N_w * K^{(M)})]$, where T denotes the number of total iterations, N_d, N_e and N_w represent the number of documents, keyword-document pairs and keywords respectively. Such time complexity stems from the computation of factors $\psi_{ij}(h_{ij})$, $\phi_{ij}(y_i, h_{ij})$ and $\tau_i(y_i)$ given current model parameters $\boldsymbol{\theta}$. After computing the factors, the time complexity of belief propagation is $O(N_e)$, and updating parameter $\boldsymbol{\theta}$ takes $O(|\boldsymbol{\theta}|)$, which are much faster than the factor computation and could be ignored.

Extension to Ranking Problem

This section discusses how to extend T-RBM to ranking problems, where training labels are ordinal instead of binary, *e.g.*, $\hat{y}_i \in \{0, 1, 2\}$, denoting {"irrelevant", "relevant", "perfectly relevant"}.

As the solution, we apply the cumulative logits approach [45] to convert the ranking problem back to a binary classification problem. Assume that each $\hat{y}_i \in \{0, 1, ..., V\}$, we will construct V different binary document classifier separately. For the v-th document classifier, we partition the data into two groups:

$\{y_i < v\}$ and $\{y_i \geq v\}$, and learn a document classifier $P(y_i \geq v|d_i, \mathbf{w_i}, t_i)$ based on the learning algorithm in Section 3.4.3. Given the results of the classifiers, we can compute the expected relevance as the ranking score of each document, given as follows,

$$\mathcal{F}(d_i, t_i) = \sum_{v=1}^{V} v * P(y_i = v|d_i, \mathbf{w_i}, t_i) \tag{3.21}$$

$$= \sum_{v=1}^{V} v * [P(y_i \geq v|d_i, \mathbf{w_i}, t_i) - P(y_i \geq v-1|d_i, \mathbf{w_i}, t_i)] \tag{3.22}$$

3.5 Experiment

In this section, we compare the overall performance of T-RBM with four state-of-the-art baselines, to demonstrate the effectiveness of applying T-RBM for cross-task document scoring problems.

3.5.1 Experiment Setting

To demonstrate the capacity of T-RBM on general cross-task document scoring problems, we study three different applications – verbose-query-based retrieval, entity-centric document filtering and cross-domain sentiment analysis – on different datasets with specification shown in Figure 3.4.

1. *Robust (Verbose-query-based Retrieval).* In order to construct a large dataset, we combined two newswire collections released by TREC 2004 and 2005 Robust tracks. Unlike most datasets that contain only short keyword queries, these two collections are the largest publicly available datasets that have detailed query descriptions, which could be used as verbose queries. Following previous verbose query works [8], we only used the $\langle desc \rangle$ portions of TREC queries, and ignored the $\langle title \rangle$ portions. Given one query, each document is labeled as 2–perfectly relevant, 1–relevant and 0–irrelevant.

2. *TREC-KBA (Entity-centric Document Filtering).* This dataset includes 29 Wikipedia entities covering living persons from different domains and a few organizations. For each entity, 100∼3000 candidate documents are collected and labeled as garbage, neutral, relevant or central. Following the same procedure in the previous work [80], we got binary labels by viewing central and relevant documents as positive, and others negative.

3. *Review (Cross-domain Sentiment Analysis).* This dataset was constructed by Blitzer *et al.* [10] through selecting Amazon product reviews from four different domains: books, DVDs, electronics and kitchen appliances. Each domain contains 1000 positive, 1000 negative and 3000∼5000 unlabeled reviews. We selected 60 pivot keywords by mutual information as suggested by Blitzer *et al.* [10], and learned pivot predictors

Dataset	*Num of Tasks*	*Num of Docs*	*Num of Positive Docs*	*Classification / Ranking*
Robust	249 Queries	349,093	Perfectly Relevant: 3,821 Relevant: 20,147	Ranking
TREC-KBA	29 Entities	52,238	24,704	Classification
Review	4 Domains	8,000	4000	Classification

Figure 3.4: Dataset specification.

(for SCL) and Pearson Correlation (for T-RBM) based on unlabeled reviews.

Among the three applications, verbose-query-based retrieval was studied as a ranking problem in previous works, while entity-centric document filtering and cross-domain sentiment analysis were treated as classification problems. Following such conventions, we used ranking-oriented metrics like NDCG@k and MAP (Mean Average Precision) for verbose-query-based retrieval, and classification-oriented metrics including precision, recall, F1-measure and accuracy for the other two tasks.

To confirm the confidence of the comparison, for both verbose-query-based retrieval and entity-centric document filtering, we divided queries into 5 sets, and adopted 5-fold cross validation with standard t-test. For cross-domain sentiment analysis, we applied the evaluation method in [10], which trained one model for each domain, adapted the model to the other three domains and reported the average ranking performance over 12 sets of results.

To deal with unbalanced data, we reweighed the training instances to balance positive and negative data. All features were first standardized to reduce the impact of different feature scales. In those baselines which require classifier or ranker learning, we used SVMLight [34] and RankSVM [36] to learn the feature weightings with default parameters. In T-RBM, we empirically set the number of keyword importance level L to be 2 and regularization factor λ to be 0.005, and will later investigate their impact.

3.5.2 Quantitative Evaluation

Overall Performance

We designed different baselines to incrementally validate the performance of our proposed T-RBM model. First, we experimented the StandardLearner baseline to demonstrate the necessity of realizing learning to score keywords. Second, we compared T-RBM with LinearMapping [80, 8, 44] and BoostMapping [80] to verify the effectiveness of T-RBM on learning to adapt keyword contribution. In cross-domain sentiment analysis, we also compared T-RBM with SCL [10] to demonstrate that T-RBM well realizes the domain adaptation insight proposed by Blizter *et al.* [10].

1. *Standard Learning Model* (StandardLearner). This baseline solves the problems by standard learning to

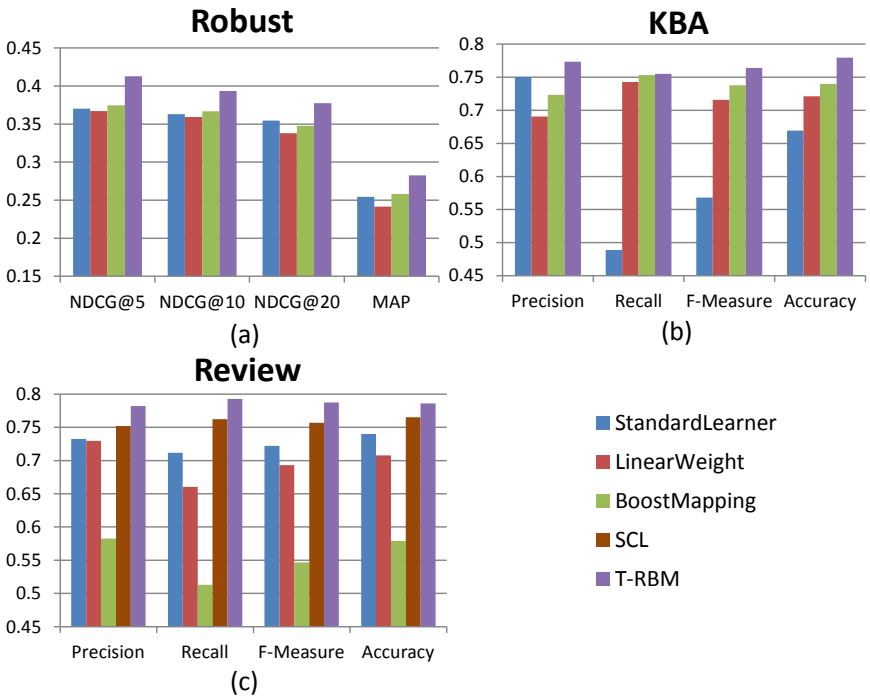


Figure 3.5: Performance comparison with baselines.

rank/classify techniques, which defines the scorer by Eq. 3.4. In the feature design, for verbose-query-based retrieval, we followed the LETOR benchmark [48] to extract ranking features such as TFIDF, BM25 and Language model for each query-document pair; for entity-centric document filtering, as the input query is an entity document, we designed features to calculate the document similarity using different metrics introduced in [31], *e.g.*, Euclidean distance, cosine similarity; in cross-domain sentiment analysis, we used all keywords as features and disregarded domain differences.

2. *Linear Weighting Model* (LinearWeight). As discussed in Section 3.4.1, the models proposed by Bendersky *et al.* [8], Lease *et al.* [44], and Zhou *et al.* [80] all belong to the same category. This baseline extends such models to support multiple intra-features, which defines the scorer by Eq. 3.8.

3. *Boosting Mapping Model* (BoostMapping). This baseline adopts the boosting framework proposed by Zhou *et al.* [80], which defines the document scorer by Eq. 3.9. Since learning keyword clusters based on multiple intra-features is difficult, we first learned the keyword clusters c_i based on only $DocTF(w, d)$, and then applied the SVM or RankSVM again to re-learn the feature weightings γ_{ij}.

4. *Structural Correspondence Learning* (SCL) [10]. This baseline implements the SCL algorithm (introduced in Section 3.2) proposed by Blitzer *et al.* We used SVMLight to train pivot predictors and the final classifiers, and adopted the same parameter setting used in [10].

Figure 3.5 demonstrates T-RBM consistently outperforms other baselines in three applications. Specif-

ically, we observe that T-RBM achieves encouraging improvement against runner-up BoostMapping in verbose-query-based retrieval (NDCG@5 +10.2%, NDCG@10 +7.3%, NDCG@20 +8.5% and MAP +9.4%) and entity-centric document filtering (F-Measure +3.5% and accuracy +5.4%) with p-value$<$ 0.05, while in cross-domain sentiment analysis, T-RBM outperforms runner-up SCL (F-Measure +4.4% and accuracy +2.7%). We analyze the performance differences between T-RBM and other baselines in the following.

First, for the StandardLearner baseline, its performance largely depends on the quality of the adopted features. The results in Figure 3.5 show that, intuitive feature designs fail to achieve satisfactory results for general cross-document scoring problems. Specifically, in verbose-query-based retrieval, using traditional document scorers as features does not perform well because of the existence of noisy keywords. In entity-centric document filtering, since there exist many relevant documents that are not similar to the identification page (*e.g.*, they might discuss only one or two aspects of the entity), StandardLearner based on document similarity fails as well. In cross-domain sentiment analysis, the result confirms the necessity of adapting keyword importance for different domains.

Second, LinearWeight does not perform well. As we discussed in Section 3.4.1, the linear definition of keyword contribution function makes it vulnerable to noisy keywords. From the result, we can observe that LinearWeight achieves relatively better performance in verbose-query-based retrieval, which is a task that involves less noisy keywords compared with the other two. The results confirm the importance of fulfilling the inferred sparsity requirement.

Third, BoostMapping fulfills the inferred sparsity requirement, by eliminating the contribution of keywords from noisy clusters; however, it might easily overfit the dataset in some specific settings. In the experiment, we can observe that BoostMapping outperforms LinearWeight over Robust and KBA datasets, but achieves very poor performance on the Review dataset. That is because, unlike the first two applications where the training dataset contains many different tasks (*i.e.*, queries or entities), in cross-domain sentiment analysis, the training data only contains one domain at a time. As the result, BoostMapping severely over-fits the training domain and fails to be generalized to the other domains.

Fourth, SCL outperforms other baselines, and the result is consistent with the performance reported in [10]. However, SCL relies on the idea of pivot predictors to realize domain adaptation, which could not be generalized to other cross-task document scoring problems that do not have the concept of pivot keywords. Moreover, in the experiment, we discover that the performance of SCL is sensitive to the model used for training the pivot predictors – different pivot predictors (*e.g.*, trained by SVM and logistic regression) tend to result in very different prediction performance.

Finally, T-RBM outperforms all the four baselines on three different applications. Essentially, the insight

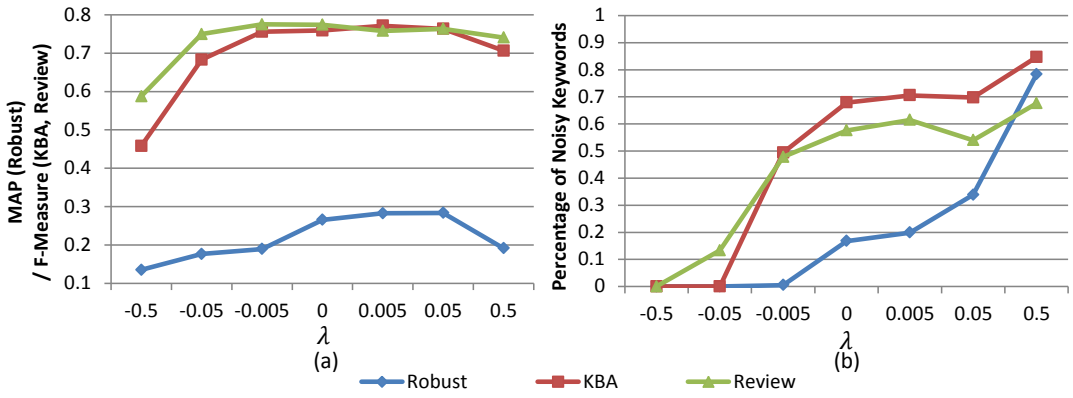


Figure 3.6: Effect of different parameters.

of T-RBM is very similar to BoostMapping, which classifies keywords into groups based on their meta-features, and learns the importance of different groups. However, different from BoostMapping which adopts a greedy algorithm to generate clusters, T-RBM takes advantage of Markov Network to jointly learn the weightings for meta-features and intra-features, which greatly reduces the risk of over-fitting. The results demonstrate that T-RBM achieves satisfactory performance on both learning to rank and domain adaptation.

Different Parameters

In T-RBM, we have to manually determine two parameters: the number of keyword importance levels L and regularization parameter λ. We experimented different values of L ranging from 1 to 5, and discovered that the performance of T-RBM is not sensitive to L. That is because, as we discussed in Section 3.4.3, even if we set $L = 1$, T-RBM can still model different levels of keyword importance by the marginalization of hidden variables.

In this section, we only investigate the impact of λ. Figure 3.6 shows how different values of regularization parameter λ affect the performances of T-RBM (in Figure 3.6 (a)) and the number of keywords that are classified as noise (in Figure 3.6 (b)). In Figure 3.6 (a), we can discover that, when λ is negative, the performance of T-RBM decreases significantly for all three datasets. That is because, as Figure 3.6 (b) illustrates, a negative value of λ will tempt the model to judge more keywords as important, making the scorer vulnerable to noisy keywords. The result again confirms the importance of fulfilling the requirement of inferred sparsity. T-RBM usually achieves the best performance when λ is around 0.005 and 0.05. Continually increasing the value of λ will over-regularize the model and lower down the performance.

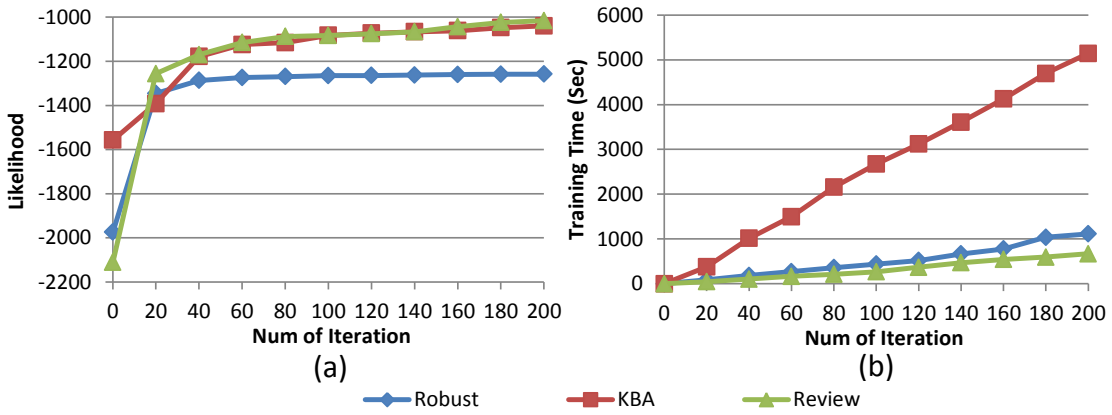


Figure 3.7: Learning efficiency.

Data set	*Task*	*Unimportant (L=0)*	*Important (L=1)*
Robust 2005	Query: Marine Vegetation	as, of, purpose	Drug purpose, marine vegetation, harvest
	Query: Abuse of E-Mail	by, in, through	prevent excess, many people, mail
KBA	Entity: Jim Steyer	2011, http, media	privacy, manager, expert, professor
	Entity: Douglas Carswell	http, UK, time	donate, foundation, enthusiast, localist
Review	Domain: DVD	was, you, just	amazing, great performance, great dvd
	Domain: electronics	what, get	inexpensive, works great, fit into, faster

(a) Examples of important/unimportant keywords.

Data set	*Meta-Feature ($\theta_{k1}^{(M)}$)*	*Intra-Feature ($\theta_{k11}^{(I)} - \theta_{k10}^{(I)}$)*
Robust 2005	IDF[Uni](1.118), IDF[Bi](0.535), QueryTF [Bi](0.301), IsVerb[Uni](0.209)	DocTF_Log[Uni](0.744), HeadTF_Log [Uni](0.178), WindowTF_Raw[Bi](0.103)
KBA	QueryPos (-0.829), IsNum(-0.353), IDF (0.169), TextTF (0.082), TFInInfoBox(0.075)	DocTF_Noramlized(0.082), TitleTF_Log (0.034) AnchorTF_Raw(-0.072)
Review	Cor[a_wonerful](14.662), Cor [easy_to](9.097), Cor[poorly] (-7.746), Cor[enjoable](7.694)	DocTF_Raw(0.0362), DocTF_Log(0.023) DocTF_Normalized(0.005)

(b) Features with highest absolute weightings.

Figure 3.8: Case study.

Training Efficiency

In this section, we are going to investigate the training efficiency of T-RBM. Figure 3.7(a) shows that the likelihood value converges very quickly within around 100 iterations in all three applications, which, as Figure 3.7(b) displays, takes 358 seconds for verbose-query-based retrieval, 2672 seconds for entity-centric document filtering, and 211 seconds for cross-sentiment analysis. In general, the training speed is very fast to get a stable solution.

3.5.3 Case Study

To give an intuitive illustration of how T-RBM identifies important/unimportant keywords for cross-task document scoring applications, in Figure 3.8 (a), we perform case studies by showing 2 example tasks per dataset, and listing some of their unimportant ($L = 0$) and important keywords ($L = 1$) based the value

of $P(h_{ij})$ derived by T-RBM. To analyze the effect of our designed features, we list the meta-features and intra-features which have the maximal absolute weightings in Figure 3.8 (b).

As Figure 3.8 displays, in terms of meta-features, in verbose-query-based retrieval, it meets our expectation that *IDF*, for both unigrams and bigrams, has the highest weightings; the result also includes *QueryTF[Bigram]*, implying that bigrams (*e.g.*, "drug purpose," "prevent excess") usually have higher importance than unigrams. In entity-centric document filtering, *QueryPos* is the most important meta-feature, demonstrating the position of a keyword is a very useful signal for identifying important keywords from a long Wikipedia page; *TFInfoBox* also has high weighting, which verifies our intuition of using Wikipedia page structure to identify important keywords. In cross-domain sentiment analysis, the result shows that the correlation with keywords such as "a_wonderful" and "easy_to" is helpful for identifying the sentiment of a keyword, demonstrating that our framework successfully realizes the insight proposed by Blizter *et al.* [10]. In terms of intra-features, we can observe *DocTF* always has the highest weighting, whereas, different applications favor different term frequency representations. For example, in entity-centric document filtering, *DocTF_Normalized* has higher weighting because the TREC-KBA dataset contains a large number of short documents (*e.g.*, tweets), and without normalization, the scorer would severely bias towards long documents.

3.6 Framework Generalization

In this section, we discuss the generalization capability of our cross-task document scoring framework. Specifically, we will study two questions: first, what types of problems could be solved by our framework, and, second, what are the advantages of our framework in solving such types of problems.

Essentially, our framework aims at scoring a target object (*e.g.*, a document) for a certain task (*e.g.*, a query) based on their connection through some intermediate semantic units (*e.g.*, keywords) – such a relationship among target objects, tasks and semantic units is very common in many research problems. For instance, in recommendation system [69, 84], taking movie recommendation as an example, we can view a movie as a target object, recommending movies for a user as a task and movie attributes such as genres and actors as semantic units. The scores of movies for a user are determined by the connection between movies and the user through movie attributes, specifically, whether a movie has some attributes that match the user's interest. As another less obvious example, our framework could also be adopted to face recognition [79] by viewing a gallery image as a target object, finding similar gallery images for a probe image as a task and distinctive features, *e.g.*, nose and eyes, as intermediate semantic units – the similarity between a probe

image and a gallery image is measured by how they contain their shared distinctive features.

To solve these types of problems, our framework has two advantages. First, we propose a convenient feature design mechanism, which, rather than directly describes how a target object matches a task, chooses to characterize how a target object and a task are connected to intermediate semantic units (*i.e.*, meta-features and intra-features) respectively. For example, in movie recommendation, we can design meta-features to characterize how a user likes the attributes of a movie, and intra-features to represent how a movie matches those attributes. Second, we develop an inferred sparsity-based graphical model, which could effectively filter the interference of noisy semantic units. For example, from a lot of attributes of a given movie, our framework could help discover those most important attributes that affect the preference of a user and filter other attributes.

In order to demonstrate how our framework is related to and different from existing solutions, in this section, we use query expansion [16, 37, 66, 65, 42], an actively studied research problem in information retrieval, as an example. Query expansion tackles the term mismatching problem in information retrieval, *i.e.*, user queries and relevant documents do not use the same keywords. In order to increase the matching probability, different query expansion techniques are proposed. For example, interactive query refinement [37, 66] proposes to interact with users to refine queries; relevance feedback [65] suggests using the initially returned result of a given query to form a new query; the idea of search result clustering [42] tries to cluster search results, pick up keywords as labels for each cluster and use cluster labels as expanded queries.

To apply our framework, we can view a document as a target object, finding relevant documents for an input query as a task and keywords as semantic units, which is similar with entity-centric document filtering; however, as the major difference, we will use meta-features to characterize "how a keyword is expanded from the original input query" rather than "how it appears in the query." For example, to realize the pseudo relevance feedback idea [65], we can design meta-feature "how many times an expanded keyword appears in the top five initially returned results" with feature weights representing the confidence of such expanded keywords.

By applying our framework, we could conquer several challenges which traditional query expansion techniques fail to handle.

First, our framework manages to automatically learn the importance of expanded keywords. Determining expanded keywords' importance is difficult, because it is impractical to require keyword-level training labels. As the result, traditional techniques either assume that expanded keywords share the same weights with original query keywords [37], or determine their weights through manually crafted formulas [66, 65, 42], which are difficult to be extended to different data. Our framework naturally solves such a problem, as it

only requires document labels to learn the importance of keywords. Moreover, we can also use our framework to combine different query expansion techniques by designing meta-features such as "which query expansion technique generates an expanded keyword."

Second, our framework can tackle the problem of noisy expanded keywords. As reported in previous works [68], expanding a query with more keywords would not necessarily help improve the performance, but might harm the retrieval effectiveness as it brings a lot of noisy information. To handle such a problem, traditional techniques require human efforts to tune the number of expanded keywords, while as the key advantage of our framework, we can rely on the inferred sparsity property to automatically filter noisy expanded keywords.

Although our framework could easily realize different query expansion ideas through the design of meta-features, there exist ideas which our framework fails to fulfill. For example, Lv *et al.* [50] point out that the importance of expanded queries should be dynamic, which depends on not only how they are expanded, but also the discrimination of query and feedback documents – for example, the more discriminative the query is, the more drifting tolerant it could be, and it would be safe to utilize the feedback information. Our framework fails to fulfill such an idea, because it learns feature weights as constants to represent the confidence of expanded keywords, while this idea requires feature weights to be dynamic rather than constant. In our future work, we will try to summarize such requirements which our approach fails to handle, and further improve the generalization capability of our framework.

Chapter 4

Query For Entities: Data-Oriented Content Query System

4.1 Introduction

With the ever growing richness of the Web, people nowadays are no longer satisfied with finding interesting documents to read. Instead, we are becoming increasingly interested in the various fine granularity information units, *e.g.*, movie release date, book price, the typical linguistic usages of certain phrases, *etc.*, which appear within the content of Web documents. Indeed, often our information needs boil down to looking for small pieces of information embedded in documents. Towards exploiting such rich data on the Web, we witness several emerging Web-based search applications:

Web-based Information Extraction (WIE) Information extraction, with the aim to identify information systematically, has also naturally turned to Web-based, for harvesting the numerous "facts" online—*e.g.*, to assemble a table of *all* the ⟨country, capital⟩ pairs (say, ⟨France, Paris⟩). Recent WIE efforts (*e.g.*, [23, 14, 52]) have mostly relied on phrase patterns (*e.g.*, "X is the capital of Y") for large scale extraction. Such simple patterns, when coupled with the richness and redundancy of the Web, can be very useful in scraping millions or even billions of facts from the Web.

Typed-Entity Search (TES) As the Web hosts all sorts of data, several efforts (*e.g.*, [17, 13, 19, 12]) proposed to target search at specific *types* of entities, such as *person* names near "invent" and "television." Such techniques often rely on readily available information extraction tools to first extract data types of interest, and then matching the extracted information units with the specified keywords based on some proximity patterns.

Web-based Question Answering (WQA) Many recent efforts (*e.g.*, [11, 47, 75]) exploited the diversity of the Web for virtually any ad-hoc questions, and leveraged the abundance to find answers by simple statistical measures (instead of complex language analysis). Given a question (*e.g.*, "where is the Louvre Museum located?"), WQA needs to find information of certain type (a location) near some keywords ("louvre museum"), and examine numerous evidences (say, counting co-occurrences) to find potential answers.

With so many ad-hoc efforts exploiting Web contents, such as WIE, TES, and WQA, there is a pressing need to distill their essential capabilities—We thus propose the concept of *Data-oriented Content Query System* (or DoCQS[1]), for generally supporting "content querying" for finding data over the Web. To motivate, we observe that, as their targets, those new applications all share one key objective—to search into the *content* on the Web to explore its rich data of all kinds—much beyond finding *pages* as in traditional Web search. As their functional requirements, these applications can be distilled into the following key capabilities:

Extensible Data Types: With their focus on fine grained information, all such applications target at data entities of various types. As different applications will need different types, a general system must support type extensibility, to build specialized data types upon existing ones, in a declarative manner. *E.g.*, we can "specialize" #number into #zipcode, #population, or #price in an online fashion. Note that, to distinguish from keywords, we prefix # for data types.

Flexible Contextual Patterns: All these applications recognize desired data by its surrounding textual patterns. While phrase patterns are useful, it is limited to scenarios where sequential words can be completely specified. A general system should utilize all the available information that appears in the context of target information. Thus, the ability to support flexible, expressive contextual patterns, beyond simple phrases, is mandatory.

Customizable Scoring: To find target data, all these applications perform scoring on candidate answers, in their specific ways. A general system must support the customization of rank scoring to meet various domains, in the following aspects:

- *Weighting*: As evidences come from matching various patterns or multiple rules, we must be able to weigh those evidences to favor more confident ones.
- *Aggregation*: To account for evidences from everywhere—by the immense redundancy of the Web—it is crucial to aggregate information. We thus need customizable aggregation to apply different strategies for different applications.
- *Ranking*: Essential in any search system, we need ranking for presenting results in an ordered manner so that the most promising results appear at top places.

For these capabilities, we must generalize their support from current limited realization in various ad-hoc efforts, which rely on a restrictive set of fixed data types, simplistic phrase patterns, and hard-coded scoring functions. We thus define our DoCQS proposal:

[1]DoCQS is pronounced as dokis.

DoCQS Definition: A *Data-oriented Content Query System* supports users, with respect to a corpus of documents such as the Web, to use keywords or data types to query for relevant values of their desired data types in the contents of the corpus, by specifying flexible patterns and customizing scoring functions.

To realize a general DoCQS, this chapter proposes its corresponding content query language, CQL, and presents the underlying indexing and processing framework, with the following contributions:

- We propose the concept of a general content query system, by distilling its essential requirements.
- We design a flexible query language CQL for content querying.
- We develop indexing design and query processing for the efficient support of DoCQS.
- We validate with extensive experiments over *realistic Web corpora* in concrete applications.

4.2 Related Work

Towards searching fine granularity data, there are several recent systems (*e.g.*, [17, 19, 38, 13]). Chakarabarti et al. [17] propose to search for annotations using proximity in documents. *EntityRank* [19] proposes the problem of entity search, and studies a probabilistic ranking model. NAGA [38] builds a knowledge base over extracted entities and relationships and supports flexible query over it. All these works leverage off-the-shelf extraction tools for extracting entities from text, and thus support only a fixed set of data types. Our DoCQS provides a framework for type extensibility— specializing basic data types into specific data types in an ad-hoc manner, and therefore greatly extending the application scenarios.

Our work aims at designing a language and framework to support extracting and querying of fine grained data holistically over the entire corpus. Some recent works (*e.g.*, [62]) address "declarative" mechanisms for information extraction, proposing languages or algebra to specify information extraction. We note that these attempts take a *document-based* approach, in that their operations apply to one document at a time. Our framework performs extraction and search *holistically* over the whole corpus.

In implementation, in terms of our indexing techniques, our work is related with existing work on using inverted index for efficient entity extraction. Our proposal is essentially supporting SQL-like DB queries, over IR type of indexes. Ramakrishnan et.al [61] propose to use inverted index to support regular expression extraction. Agrawal et al. [3] study the problem of large-scale extraction of dictionary based entities using keyword inverted index. Our work goes beyond traditional inverted index, by building specialized indexes between keywords and data types. Therefore, as we show in Section 4.6, our solution achieves much better efficiency compared to such standard inverted index approaches. BE engine [14] proposes "neighborhood index" to efficiently support extraction based on phrase patterns. As the name indicates, it is limited to

only extraction based on phrase patterns. Our DoCQS is flexible in supporting a wide variety of context operators, thus allowing extending new data types from basic data types. Further, our experiments also show that our indexing is more efficient than the neighborhood index.

4.3 Data Model

With the search target of various data types inside documents, the traditional IR data models used (*e.g.*, the vector space model which views query and documents as keyword vectors) are no longer appropriate. We need to come up with a new data model, to meet the various capabilities of a content query system (Section 4.1).

First, our search target is data types. Each type captures a domain of values (*e.g.*, prices). In the contents of a Web corpus, each value has multiple occurrences. To query data types, we need to record each occurrence with the corresponding instance value, document, and position—*i.e.*, a "tuple" of occurrence information. We can naturally represent such tuples in a relational table.

Second, the content query system calls for flexible pattern matching, as well as customizable scoring of weighting, aggregation, and ranking. These operations go much beyond the standard IR operations of containment check and relevance-based scoring of individual documents. In fact, pattern constraints, aggregation, and ordering are concepts more widely used in relational operations.

Indeed, the content-querying capabilities consistently call for a relational model, which supports the modeling of data types as relations, and the various relational operations for pattern matching, aggregation, and ranking (WHERE, GROUP BY, ORDER BY). Thus, taking relational modeling, we conceptualize the E-R diagram for the entities and relationships in our DoCQS in Figure 4.1.

In this framework, there are three types of entities in the E-R diagram: document D, built-in data type T_i (for the set of built-in data types, *e.g.*, #number, #organization) , and keyword K_j (for all keywords, *e.g.*, "Chicago"). Note that we can view both T_i and K_j as *Is-A* data type—since we can consider a keyword as a data type with a specific literal value (*e.g.*, "Chicago"). Each document can be identified by its unique document ID. Each *occurrence* of a data type (*e.g.*, #person) can be identified by a unique *ID*, its specific instance *value* (*e.g.*, "Bill Smith"), *confidence* (*e.g.*, 0.9 probability), *span* (*e.g.*, a span of 2 words).

As relationship, a data type occurs at the content of some document, at a certain position *pos*. Each *occurrence* is unique and can only appear at one position of a document. Therefore, the *Occurs-At* relationship between data type and document is *many-to-one*. As mentioned earlier, keyword is simply a special data type, whose value is always the keyword itself with a span of 1 .

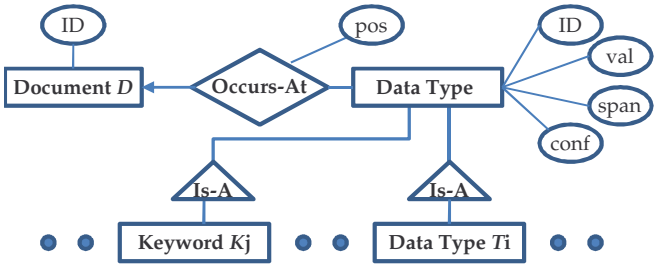


Figure 4.1: ER diagram for the proposed content query system.

With the ER diagram outlined, we are ready to come up with the schema of our relational model. We notice, with the many-to-one relationship between data type and document, we can combine the two entities into one relation. Moreover, we also realize that a specific document *ID* with a position pos can uniquely define an occurrence of a data type, therefore eliminating the need of the ID attribute of the data type. We can thus define the schema for two types of relations (keyword and data type) as follows:

- *doc*: The document ID where the keyword/data type appears.
- *pos*: The word position of the occurrence.
- *span*: The number of keywords that the occurrence covers. For keyword, span is always 1.
- *val*: The content of the occurrence. For keyword, the content is always itself (and therefore omitted).
- *conf*: The confidence, which measures the probability that the data occurrence belongs to its data type. As data extraction is inherently imperfect, the confidence may not be 100%.

We note that the two types of relations will materialize to many tables: In the overall schema, we are dealing with M data type relations noted as $T_1, T_2, \ldots, T_M$, as well as N keyword relations noted as $K_1, K_2, \ldots, K_N$. Figure 4.2 shows an example in turning a text corpus into our relational model, with two concrete relations illustrated, one for keyword "population" and one for #number.

4.4 Content Query Language (CQL)

This section will discuss our design of the Content Query Language (CQL) to serve the need of DoCQS. We will first reason the general form of the query language, based on the capabilities needed and the relational model derived. Then we will present the overall general specification of the query language, followed by in-depth discussion of each component of the query language.

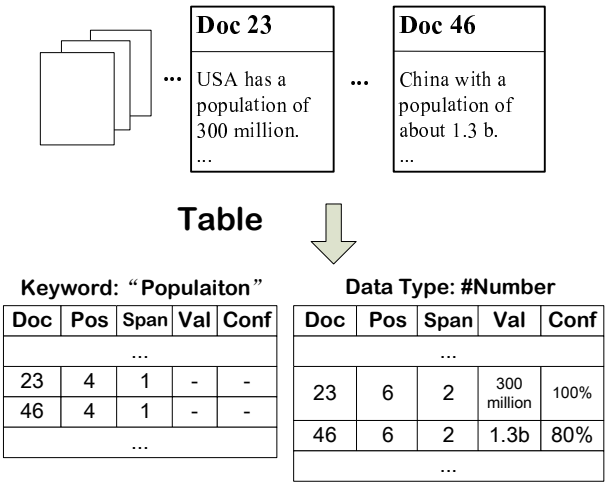


Figure 4.2: Data model: Relations for keywords and data types.

4.4.1 Design Principle

With the relational model chosen as our data model in Section 4.3, we now discuss the design of our query language based on the relational model.

While SQL is the widely accepted query language over relational model, with the special need of a content query system, our goal of the design is to customize the query language to meet the various requirements of DoCQS. Therefore, we propose our own query language CQL, which takes a special form of SQL with several new constructs for the need of querying data types inside document content, with examples shown in Figure 4.3.

Based on the ER diagram in Figure 4.1, a CQL statement targets at retrieving tuples from the relational tables, which are transformed from the text corpus as shown in Figure 4.2. Specifically, each retrieval statement involves several steps: specifying source tables, joining tables, filtering tuples, aggregating, and finally ranking results. Consider the example query **Q1** in Figure 4.3(a), which aims at deriving ⟨location, population⟩ pairs from documents. It first joins the #location and #number tables by document ID, utilizes conditions (*e.g.*, a sequential pattern {#location has population of ?⟨0,3⟩ #number}) to retrieve ⟨location, population⟩ pairs, aggregates those pairs to estimate the confidence based on the redundancy of potentially correct answers, and finally ranks the results by confidence. To customize for these operations, CQL takes a special form of **SELECT** ... **FROM** ... **WHERE** ... **GROUP BY** ... **ORDER BY** ... . We now discuss in detail the functions of each clause.

First, in the SELECT clause, we are interested in retrieving relevant *values* of target data types. This is supported by having the SELECT clause to project onto the *val* attribute of data types.

Second, in the FROM clause we specify all the participating keyword and data type relations for the

```
SELECT #location.val, #number.val
FROM #location, #number
WHERE                                   //w1 = 0.95, w2 = 0.8
      (P1=pattern("{#location has population of ?⟨0, 3⟩
            #number}")∧
      P2=∼LikeLargeNum(#number))
      WITH P1.score ∗ P2.score ∗ w1
      ∨
      (P3=pattern("[#location population #number]⟨6⟩")∧
      P4=∼LikeLargeNum(#number))
      WITH P3.score∗P4.score∗w2
GROUP BY #location, #number
      WITH 1 − ∏(1 − conf)
ORDER BY conf ()
```

(a) Query **Q1**: Data retrieval with explicit-scoring.

```
SELECT #location.val, #number.val
FROM #location, #number
WHERE (                                 //w1 = 0:95, w2 = 0:8
      pattern("{#location has population of ?⟨0, 3⟩
      #number")[w1]
      OR pattern("[#location population #number] ⟨6⟩")[w2]
   )AND ~LikeLargeNum(#number)
GROUP BY #location, #number
ORDER BY conf()
```

(b) Query **Q2**: Data retrieval with implicit-scoring.

```
DEFINE DATATYPE #GDP AS #number
WHERE pattern("[GDP #number] ⟨10⟩")
AND ~LikeLargeNum(#number)
```

(c) Query **Q3**: Data type definition.

Figure 4.3: Some example CQL queries.

query. These are the basic relations upon which our query operates.

Third, our WHERE clause will support fuzzy text oriented patterns. We will introduce our own intuitive pattern syntax (*e.g.*, "*pattern*([professor #person]⟨10⟩)") to hide the detailed relational data operations in the background.

In addition to the standard SQL constructs, with the need to differentiate the importance of different patterns, as well as the need to judge how well each pattern is matched, our WHERE clause will accommodate a weighting scheme of the patterns, in addition to the specification of a series of text-oriented patterns.

Moreover, with the need to define specialized data types based on existing data types, our query language offers a special data type definition construct, which is inspired by the "view" concept in databases.

Finally, CQL supports a plugin function library, with a set of functions provided by default. The library contains various text retrieval measures such as term frequency, PageRank, *etc.* Such functions could be utilized for flexible customization in weighting and ordering.

4.4.2 CQL Specification

We now introduce the general form of our content query language. Similar to the data retrieval and manipulation aspects in SQL, we define two types of operations in CQL, data retrieval operation and data type definition. We now describe their syntax.

Data Retrieval Operation To retrieve relevant data values of data types $T_1, \ldots, T_n$.

SELECT $T_1[.*], \ldots, T_n[.*]$

FROM $T_1, \ldots, T_n$

WHERE *condition*

[**GROUP BY** $T_k[.val]$ [**WITH** $GS(conf)$]]

[**ORDER BY** *expr*]

where *condition* in the WHERE clause can take the following two forms:

1) Explicit-scoring condition

condition:=$pattern_1 \wedge \ldots \wedge pattern_l$ **WITH**

$LS(pattern_1.score, \ldots, pattern_l.score)$

$(\vee$ *condition*$)^*$

2) Implicit-scoring condition

condition:=*pattern*$[w]$ ({**AND**|**OR**} *condition*)*

In the above specification, T_i refers to a specific data type and $T_i.*$ indicates $\langle T_i.doc, T_i.pos, T_i.span, T_i.val, T_i.conf\rangle$. If no attribute is specified, "*.val*" is used for default because we are interested in the value of the data type in most cases. The FROM clause lists all the data-type tables involved; it omits the keyword tables (for simplicity) since their values are not of interest. The data type relations in the FROM clause are natural-joined by the *doc* attribute. For conciseness we use "," instead of $\bowtie_{doc}$.

To meet the query demands of content query tasks, the data retrieval operation allows users to customize their scoring function in the WHERE clause for measuring how well the given patterns are matched over individual documents, and in the GROUP BY clause for measuring how frequent results appear over the data corpus. To accommodate the different semantics of these two measures, the query language supports two scoring functions: *LS* in the WHERE clause for local scoring within a document, and *GS* in the GROUP BY clause for global scoring across documents.

By default, for users who do not want to specify detailed scoring functions, we support the implicit-scoring condition, which eliminates the need of explicit specification of the *LS* function in the WHERE clause and the *GS* function in the GROUP BY clause. In this case, the default *LS* and *GS* functions are implicitly applied. We will discuss this aspect in more details in Section 4.4.4.

Data Type Definition To derive specialized data type T_{new} from an existing base data type T_{base}.

DEFINE DATATYPE T_{new} **AS** T_{base}

WHERE *condition*

The data type definition is used for extending basic data types into special data types, such as defining

#population from base type #number, as we motivated in Section 4.1. Derived data types will have the exact same schema as the basic data types, and can therefore be seamlessly used in the same way as the basic data types for retrieval. As the above syntax shows, we currently limit the definition to be based on one existing data type only, which captures most cases in practice.

With the general specification of CQL outlined, we now discuss the special constructs in the language that are tailored to the need of a data-oriented content query system. Specifically, we will zoom into the specification and weighting of patterns, the customization of scoring functions, and the definition of new data types.

4.4.3 Pattern & Weighting

The **WHERE** clause is used to select those tuples satisfying the indicated conditions. This section first introduces the specification of the pattern conditions, followed by the weighting and scoring scheme of conditions.

Pattern conditions play a key role in selecting desired results. For example, we can utilize pattern conditions to select numbers following phrase "population of", or person names near keywords "professor" or "doctor." By hiding relational operations (*e.g.*, joining a series of keyword and data type tables by some conditions) from users, these pattern conditions provide an intuitive interface for users to describe flexible patterns. In our framework, we support four basic pattern conditions, and they can be further extended for different application demands.

• **Sequential Pattern**

Syntax: $\{X_1\ X_2\ ...\ X_n\}$

Matched when keywords or data types X_i appear in sequence (*e.g.*, {#location has population of ?⟨0,3⟩ #number} in Figure 4.3(a)). It is defined as:

$$\sigma_{\bigwedge_i (X_i.pos+X_i.span=X_{i+1}.pos)}(X)$$

where a *term* X_i stands for any keyword or data type relation, and X is shorthand for their natual joins, *i.e.*,

$$X_1 \underset{X_1.doc=X_2.doc}{\bowtie} X_2 \ldots X_n.$$

Notice, X_i can also be a nested pattern relation, which allows the combination of different patterns. Wildcard is also allowed in the sequential pattern. In the example above, the wildcard ?⟨0,3⟩ allows 0 to 3 words (*e.g.*, "almost", "close to") between "of" and #number. In such cases, the above position constraint would become

$lower \leq X_{i+1}.pos - (X_i.pos + X_i.span) \leq upper$, when wildcard $?\langle lower, upper\rangle$ is inserted between X_i and X_{i+1}.

• Window Pattern

Syntax: $[X_1, X_2, ..., X_n]\langle m\rangle$

Matched if all keywords/data types/patterns appear within an m-words window. It is defined as:

$$\sigma_{\bigwedge_{i,j}(X_i.pos+X_i.span-X_j.pos<=m)}(X)$$

Notice, a pattern could also be nested in other patterns, for example, [#number {United States}]⟨10⟩. In this case, X_2 is a pattern whose *pos* and *span* is not readily available. Online computation will generate $X_2.pos$ as United.*pos* and $X_2.span$ as (States.*pos* + States.*span* - United.*pos*). Finally, the conditions of the main pattern and nested patterns are connected by conjunction.

• Disjunction Pattern

Syntax: $(X_1|X_2|...|X_n)$

Matched if any keyword/data type/pattern in the list is matched, for example, {#number (people|inhabitant)}. It is defined as:

$$X_1 \cup X_2 \cup ... \cup X_n$$

• Inner Pattern

Syntax: $(X_o : X_i)$

Matched if X_o is matched and X_i lies in the scope of X_o, for example, pattern "(#organization:university)" returns #organization instances that contain the "university" keyword. It is defined as:

$$\sigma_{X_i.pos>=X_o.pos \bigwedge X_i.pos+X_i.span<=X_o.pos+X_o.span}(X)$$

These four basic pattern conditions are designed based on common content query demands. Sequential patterns focus on extracting data types surrounded by specific phrase context words (*e.g.*, numbers following phrase "population of" are highly possible to be of the population data type). Window patterns could be used to locate instances of data types of certain topics (*e.g.*, finding the inventor of TV near words "television", "invention"). Disjunction patterns allow data types to be generated by different patterns. Inner patterns can retrieve data types with specific content (*e.g.*, a #location containing "university" is likely to be the

#university type).

In addition to the pattern conditions, which can be viewed as pre-defined Boolean functions, we also allow other user-defined Boolean or fuzzy functions in the WHERE clause. The difference between Boolean and fuzzy functions is that the former filters the result by its returned Boolean value, while the latter does not perform filtering but instead assigns scores to tuples by how well the condition is matched. For instance, we can use a Boolean function *IsYear*(#number) to select year-formatted numbers or use a fuzzy function ∼*LikeLargeNum*(#number) to favor large numbers.

Weighting of conditions is a new concept, which does not exist in SQL. Its function is to obtain the weights of conditions specified in the WHERE clause. First, we need to distinguish the importance of different patterns (noted by w_j), as we may have more confidence in one pattern condition over another. Second, we need to examine how well the pattern is matched (noted by $P_i.score$ for pattern P_i). For Boolean functions, the score is either 0 or 1, while for fuzzy functions, the score is a real value between 0 and 1 reflecting the matching degree. These two factors contribute to the final weight of a pattern.

Query **Q1** in Figure 4.3(a) shows how to customize the weighting of patterns in detail. First, the w_j parameter allows us to give different weights to different patterns. For instance, we can set w_1, w_2 to be 0.95, 0.8 for two conjunctive patterns $P_1 \wedge P_2$ and $P_3 \wedge P_4$ respectively (where a stronger conjunctive condition like $P_1 \wedge P_2$ gets higher weight). Then $P_i.score$ captures how well a pattern P_i is matched. For instance, $P_1.score$ outputs 1 if P_1 is matched, and 0 otherwise. For P_2, which is a fuzzy pattern, the score is a probabilistic value measuring the likelihood of a large number (*e.g.*, if the number value is greater than 1 billion, $P_2.score = 1$; if it is less than 1 billion more than 1 million, $p_2.score = 0.8$; ...).

4.4.4 Scoring Specification

As we have hinted briefly, there are two levels of scoring specification in our CQL.

First, local scoring LS is in charge of judging the local score of a specific tuple occurrence within a document. It takes as input the matching scores of patterns, and we can assigns weights to the patterns by multiplying their scores and the weights w_j in the formula. In addition, it can also take individual attribute values of the participating relations (*e.g.*, their *conf* values). It outputs a score *conf*, which indicates the confidence over the matched tuple occurrence. For instance, **Q1** uses $P_1.score{*}P_2.score{*}w_1$ as the local scoring function for the conjunction result of P_1 and P_2.

Second, global scoring GS is used for calculating the score of a specific group of tuple occurrences that share the same data values. This is where the aggregation comes into play, since the occurrences of the same group come from different pages and therefore their scores need to be put together. This global function

normally takes into account the local confidence scores of the matched occurrences. It can also use other common information retrieval functions (like TF-IDF, Pagerank, *etc.*). For instance, Figure 4.3(a) uses $1 - \prod(1 - conf())$ as the aggregation expression, which means that given n tuples with confidence $conf_1$, ..., $conf_n$, the aggregated confidence will be $1 - \prod_{i=1}^{n}(1-conf_i)$.

For the ease of users who do not want to fine tune the detailed scoring function, we provide implicit scoring to eliminate the need of explicit specification of the *LS* scoring function in the WHERE clause and the *GS* scoring function in the GROUP BY clause, such as query **Q2** in Figure 4.3(b). In this situation, default *LS* and *GS* functions will be applied for scoring. Specifically, the default *LS* and *GS* functions take the following forms:

- *LS* for **AND** operator: Given $(P_1$ **AND** $P_2)[w]$, the confidence of the generated result is $conf_{LS}((P_1$ **AND** $P_2)[w]) = P_1.score{*}P_2.score{*}w$.
- *LS* for **OR** operator: Given $(P_1[w_1]$ **OR** $P_2[w_2])$, the confidence is $P_1.score{*}w_1$, if the result tuple comes from matching P_1, and similar for P_2.
- Default *GS*: Given n tuples with confidence $conf_1, \ldots, conf_n$, $conf_{GS} = 1 - \prod_{i=1}^{n}(1-conf_i)$.

With the default *LS* and *GS*, query **Q2** is equivalent to query **Q1**. For conciseness, we will write CQL examples using the implicit scoring format for the rest of the chapter.

4.4.5 Data Type Definition

View, in the relational database sense, refers to a stored query, which is accessible as a virtual table. In DoCQS, we borrow this concept to define new data types over existing ones. A defined data type has the same schema as the basic data type, and can be accessed later like the basic data types with no difference. The confidence values of a new data type comes from the matching scores of various conditions. Users can customize the scoring of confidence by specifying the *LS* function in the WHERE clause.

Query **Q3** in Figure 4.3(c) defines the #GDP data type, by identifying large numbers with keyword "GDP" around them within a 10-word window. During execution, the references of new data types will be translated into the stored data type definitions and get executed accordingly. As an alternative, our system also allows the materialization of new data types and stores them on disk, which can be used directly in subsequent queries with faster response.

4.5 Indexing & Query Processing

This section discusses the indexing design, configuration and query processing of the DoCQS system. The data-oriented characteristics require a very different index architecture compared with relational tables used in traditional DBMS. In DoCQS, we mainly rely on IR style inverted indexes and present an effective index selection algorithm for query processing.

4.5.1 Indexing Design

We utilize the gist of inverted index as the essential indexing structure for the DoCQS system. Although keywords and data types are conceptually modelled as tables in our system for the convenience of defining the CQL language, we choose IR style inverted index as the underlying data structure since the most common operation of CQL is to traverse the keyword or data type tables. Sequential access of inverted index is well-known to be efficient for supporting traversal operations. For balancing processing efficiency and index space, we propose two layers of indexes in the DoCQS system: the basic inverted list layer and the advanced inverted index layer. We now zoom into these two layers.

Basic Inverted List Layer

The basic inverted list layer stores all information of keyword and data type tables defined in the data model (shown in Figure 4.2) in sequential lists. For keyword K_i, we build the traditional inverted list $I(K_i)$; while for data type T_j, its inverted list $I(T_j)$ also stores information about the other attributes (*e.g.*, *doc*, *val*) in addition to position information. Take #phone as example, its inverted list is of the form: $I(\#\mathsf{phone}) \rightarrow$ {⟨*doc*:1, *pos*:10, *val*:"123-456-7890", ... ⟩, ⟨*doc*:20, *pos*:13, *val*:"789-456-1230", ... ⟩, ... }.

With the basic inverted list layer defined above, CQL could be already executed (though not efficient, as we will improve later). Consider a simple query example **Q4**:

SELECT #number.val

FROM #number

WHERE *pattern*("{population of ?⟨0, 3⟩ #number}")

To execute the query, the DoCQS system first loads three inverted lists: I("population"), I("of"), and $I(\#\mathsf{number})$. The pointers to the three lists are incremented for checking intersecting documents. Once an intersecting document is identified, the system retrieves the postings of this document from the three lists, and joins them by the sequential pattern specified to produce the matching tuples. This execution ends when one of the lists is exhausted.

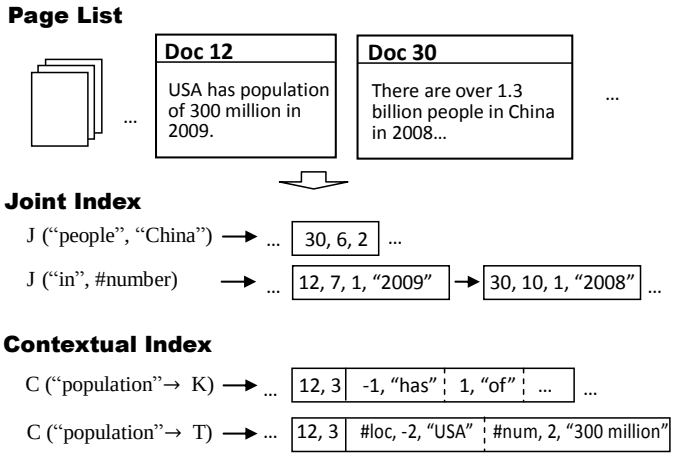


Figure 4.4: Advanced inverted index layer.

Advanced Inverted Index Layer

Although the basic inverted list layer could already support CQL execution, query performance remains a significant issue. In query **Q4**, "of" and #number appear almost in every document, which means very long inverted lists. It is very expensive to load and scan such long lists. Moreover, we can not discard such high frequency keywords (*e.g.*, "of" "such") as stop words, because they play concrete roles in describing interesting patterns. We need another index layer, the Advanced Inverted Index Layer, to further expedite query processing.

The design principle of the advanced inverted index layer in DoCQS is inspired by the index layer in databases. Traditional DBMS utilizes B+ Tree to speed up query processing using conditions in the selection clause. Consider a SQL query to retrieve the name of students who are over 18 years old. With index built for the age attribute, the system could directly get the tuples satisfying $age > 18$, without scanning through the whole table. Similarly, the basic idea of the advanced inverted index layer is to record redundant information and utilize pattern conditions to avoid traversal over the inverted list of highly frequent keywords or data types. Again consider query **Q4** in Section 4.5.1. The query execution at the basic inverted list layer is basically a join operation over I("population"), I("of"), and I(#number). If the pair ⟨"population","of"⟩ and ⟨"population",#number⟩ have been pre-joined offline, the online execution only needs to traverse these two joined-lists, which are much shorter than I("of") and I(#number), thus greatly saving the query processing time. With this insight, we propose two kinds of special inverted index structures: Joint Index and Contextual Index, with examples shown in Figure 4.4.

Joint Index The main idea of "Joint Index" is to pre-join two terms (which could be keywords or data types), and store their co-occurrence pairs within W-word window, where W could be configured by the

system. As shown in Figure 4.4, a text segment like "There are over 1.3 billion people in China in 2008" is given. With W set to 5, since the distance between keywords "people" and "China" is 2 ($< W$), we record their co-occurrence information $\langle$*doc*:30, *pos*:6, *offset*: 2$\rangle$ in the joint index J("people","China"), and we similarly construct J("in",#number). When DoCQS receives a query like "[China people]$\langle 5 \rangle$", it can use the joint index J("people","China") to greatly improve processing time because the size of the J("people","China") list is much shorter than either I("people") or I("China").

Formally, a joint index could be viewed as the pre-joining of two tables. Consider two terms X_i and X_j, whose corresponding tables (stored as inverted lists) are $I(X_i)$ and $I(X_j)$. Their joint index $J(X_i, X_j)$ is:

$$J(X_i, X_j) = I(X_i) \underset{|I(X_i).pos - I(X_j).pos| < W}{\bowtie} I(X_j)$$

For queries such as $\{X_i \ ?\langle 0, w-2 \rangle \ X_j\}$ and $[X_i, X_j]\langle w \rangle$ ($w <= W$), the query time complexity is reduced from $O(I(X_i) + I(X_j))$ (using basic inverted lists) to $O(I(X_i) \cap I(X_j))$ (using joint index).

Although such joint indexes significantly reduces query time for a large number of patterns, due to their high space cost, we can not afford building the joint index for every pair of terms in the whole corpus. The extra space cost comes from the large number of term pairs, *i.e.*, N^2 (N is the number of terms in the corpus). To alleviate the high space cost of joint index, we next propose another type of more space efficient inverted index.

Contextual Index The contextual index uses each term as the index key and stores its surrounding terms within its W-word context window. In Figure 4.4, given a text segment "USA has population of 300 million in 2009", the occurrence of "population" at $\langle$doc: 12, pos: 3$\rangle$ is recorded in its the contextual index C("population" $\rightarrow K$) (where K refers to keywords in its context), together with the information of its surrounding words (*e.g.*, $\langle$val: "has", offset: -1$\rangle$, $\langle$val: "of", offset: 1$\rangle$). For a query with sequential pattern {population of}, the system only needs to traverse C("population" $\rightarrow K$) instead of both I("population") and I("of").

Conceptually, a contextual index for term X_i is the union of a series of joint indexes sharing the term. Given term X_i, its contextual index $C(X_i \rightarrow X)$ is defined as:

$$C(X_i \rightarrow X) = \bigcup_{X_j \in X} I(X_i) \underset{|I(X_i).pos - I(X_j).pos| < W}{\bowtie} I(X_j)$$

where X represents keywords K or data types T in the W-word window context of X_i.

For queries like $\{X_i \ ?\langle$0,w-1$\rangle \ X_j\}$ where $w < W$, the time complexity is $min(O(C(X_i \rightarrow X)), O(C(X_j \rightarrow X)))$, which means the system will choose to use the shorter list between $C(X_i \rightarrow X)$ and $C(X_j \rightarrow X)$ for answering the query.

Joint Index	Description
$J(K_i, K_j)$	Keyword K_i with keyword K_j.
$J(K_i, T_j)$	Keyword K_i with data type T_j.
$J(T_i, T_j)$	Data type T_i with data type T_j.
Contextual Index	**Description**
$C(K_i \rightarrow K)$	From keyword K_i to keywords in K.
$C(T_i \rightarrow K)$	From data type T_i to keywords in K.
$C(K_i \rightarrow T)$	From keyword K_i to data types in T.
$C(T_i \rightarrow T)$	From data type T_i to data types in T.

Figure 4.5: All the index types in the advanced index layer.

Compared with joint index, the major advantage of contextual index is its relatively low space cost, because we only need one list for each term, thus at most N (the number of terms) lists. We can also leverage inverted-list compression techniques to efficiently store a term's context, since terms in the context appear in a sequential order. Empirically, we observed that, to fully build contextual indexes over a data corpus, it needs only 2-3 times the size of the standard inverted lists (with W set at 5). However, if both terms X_i and X_j have very long lists, their contextual indexes will also be long, and therefore the traversal of long lists can not be avoided.

To summarize, we show all the possible types of indexes in Figure 4.5 according to our index design. We next discuss how to configure index among these different types.

4.5.2 Index Configuration

Based on the index structure discussed above, this section introduces the detailed index configuration to achieve high query performance with reasonable space cost overhead.

To better understand the demand of query optimization in real query, we start with analyzing the actual joining time cost of inverted index lists of different length. Using the Wikipedia dataset with 3 million pages (Section 4.6), we first categorized words into three classes: high-frequency words (top 1,000 frequent words, including common words like "of", "one", "people"), low-frequency words (6,332,031 words that appear in less than 100 documents, including some rare terminology and noise words), and medium-frequency words (132,888 words, including less-common words like "CEO", "flatiron"). We then sampled 50 high-frequency words, 300 low-frequency words, and 300 medium-frequency words. For each pair of words from the sample, we performed the join of their inverted index lists to find their co-occurrences. The average join time is shown in Figure 4.6.

As we observe, from Figure 4.6, the most time consuming join operations (whose times are shown in bold-face) happen between high-frequency words with medium or high-frequency words. Nevertheless, we

	High Freq	**Medium Freq**	**Low Freq**
High Freq	**1107 ms**	**382ms**	48 ms
Medium Freq	**382ms**	20 ms	14 ms
Low Freq	48 ms	14 ms	13 ms

Figure 4.6: Average joining time cost.

also empirically observed that such combinations are used most often in query patterns (*e.g.*, in {population of ...}, "population" is medium and "of" is high-frequency terms).

Therefore, our index configuration puts priority towards optimization for such expensive joins. We notice that it is not necessary to optimize for low-frequency terms because it takes little time to join with any other terms. Mature inverted index implementations (*e.g.*, Lucene) normally support efficient document skipping, which could directly jumps to a targeted document in the inverted index during traversal. Based on this feature, a join operation between a low-frequency word (*e.g.*, 10 documents) with a high-frequency word (*e.g.*, 1 million documents) takes at most 10 random accesses over the long inverted list, which is affordable. The time statistics in Figure 4.6 also indicate that it is not crucial to optimize joins between medium-frequency and medium-frequency words.

As the insight of Figure 4.6 inspires, we choose to build joint indexes between high-frequency terms, and contextual indexes from medium-frequency terms to high-frequency terms. This choice accounts for tradeoff between time efficiency and space overhead: For joins between high-frequency terms, efficiency is the major concern. For joins between high and medium-frequency terms, space cost should be considered, due to the large number of the medium-frequency words. These requirements match the provisions of joint indexes and contextual indexes, respectively, and thus our choice.

4.5.3 Query Processing

This section discusses how to select indexes for efficient query processing. Given a pattern condition, there are many related candidate inverted lists to choose from the basic inverted list layer and the advanced inverted index layer. Consider the following condition with two patterns:

$$pattern(\text{“[population \#number]}\langle 4\rangle\text{”})$$
$$\textbf{OR } pattern(\text{“\{\#number native people\}”})$$

Assume the window size W is 4. Keywords "native" and "people" are high-frequency words while "population" is a medium-frequency word in the data corpus. There are 8 indexes related to the query as displayed in Figure 4.7. Each index has different coverage. For instance, index $C(\text{“population”} \rightarrow T)$, where T denotes all the data types, can cover "population" itself and #number in its context, since #number lies within the

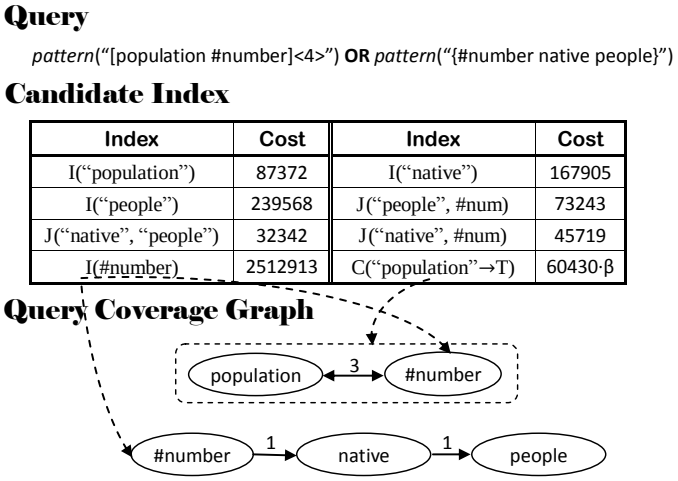


Figure 4.7: A query plan with index selection.

window of "population" in the query. Meanwhile, index $I(\#\text{number})$ can cover the #number data types for the two patterns at the same time, which means that $I(\text{"\#number"})$ could be shared across different patterns in query processing. Consider the following two index selection strategies:

- **S1**: 1) $I(\text{"population"})$; 2) $I(\#\text{number})$; 3) $J(\text{"native","people"})$.
- **S2**: 1) $C(\text{"population"} \rightarrow T)$; 2) $J(\text{"native",\#number})$;
  3) $J(\text{"native","people"})$.

Either of the strategies is sufficient for the execution of the pattern condition. Take strategy **S2** as an example. The system first traverses $C(\text{"population"} \rightarrow T)$, selects the tuple list l_1 which contains #number occurrences with "population" around satisfying the first window pattern. It then traverses and joins $J(\text{"native",\#number})$ and $J(\text{"native","people"})$ to get list l_2 containing #number occurrences satisfying the second sequential contextual pattern. Finally, the system unions l_1 and l_2 to get the final result. For this simple query, the second strategy **S2** apparently works more efficiently than **S1**, because it avoids the transversal of the long list #number. To deal with complex patterns involving a large number of data types and keywords, DoCQS needs a smart algorithm to choose an efficient index selection strategy automatically.

We model the index selection problem as a graph coverage problem. The system first transforms a query into a *Query Coverage Graph* as Figure 4.7 shows. Each node in the graph indicates a keyword or data type appearing in the query, and the directed edge from node u to v indicates that v appears after u within the distance constraint specified by the weight of the edge. Each index covers part of a graph and has different traversal cost. We define the traversal cost of an index by the number of documents included in the list. For a contextual index, we penalize the cost by multiplying a coefficient β ($\beta > 1$), since a contextual index

Algorithm 1 Index Selection Algorithm

1: Convert the given query Q into the a directed graph $\mathcal{G} = \{\mathcal{U}, \mathcal{E}\}$. Each node u indicates a keyword/data type appearing in Q. Each edge $e = \langle u, v \rangle$ in $\mathcal{E}$ indicates that for u, v appears after u satisfying certain position constraints which is recorded as the weight.
2: Candidate Index Set R. R is initialized to include all indexes
3: **for** each $u_k \in \mathcal{U}$ **do**
4: Start from u_k, DFS the graph, get an index set C_{u_k} which stores all indexes related with u_k.
5: **end for**
6: **loop**
7: $T \leftarrow R$. T is a temporary set to store indexes that could be removed.
8: **for** each C_{u_k} **do**
9: **if** C_{u_k} contains only one index i' **then**
10: Remove i' from T
11: **end if**
12: **end for**
13: **if** T is empty **then**
14: Return R as result.
15: **end if**
16: Select $i_{max} \in T$ with maximal transversal cost.
17: Remove i_{max} from R.
18: Remove i_{max} from C_{u_k}, if $i_{max} \in C_{u_k}$.
19: **end loop**

stores additional context word information. As a result, the index selection problem boils down to looking for a subset of indexes from the candidate index set to cover the whole query coverage graph with the minimal traversal cost.

Even in the simplest case, where all the indexes have the same traversal costs, this graph coverage problem is equivalent to the set cover problem, which is NP-Complete. This means an exhaustive algorithm with exponential complexity of $O(2^K)$ or $O(n^k)$ is needed for finding the optimal solution, where K is the number of candidate indexes, n the number of nodes, and k the average number of indexes that cover one node. Although many pruning conditions can be applied to shrink the search space, they can not guarantee stable performance for complex pattern conditions. For this reason, we design an $O(K^2)$ polynomial greedy algorithm.

Algorithm 1, as shown above, first builds up the query coverage graph for the pattern conditions of a given query. Second, for each node in the graph, the algorithm collects all related indexes covering the node. Third, by iteratively checking the transversal cost of each index, it drops the index list with the highest traversal cost, such that each node can still be covered by at least one index. The iteration stops when no more indexes can be removed from the candidate set. In Section 4.6, we will show that the plan selected by our algorithm is very close to that of the optimal query plan in terms of actual query processing time.

4.6 Experimental Results

This section testifies the expressiveness of CQL, and evaluates both the time and space cost of the DoCQS system. We build two datasets from two different domains, Wikipedia Text Domain and Academic Personal Homepage Domain. On each domain of data, we show how to use CQL to describe different query tasks, and demonstrate that the system could retrieve high-precision results with reasonable time and space cost.

WikiPedia Text Domain This dataset comes from the Wikipedia corpus downloaded in March 2009. We choose Wikipedia because it is an entity-rich dataset containing a lot of data type information. As the current DoCQS system focuses on unstructured text, we remove all infoboxes and tables on the pages. After data cleaning, the corpus size is 7Gb including 3 million pages. On the corpus, we target at three basic data types: number, person and location. We extracted 83 million number occurrences by a JFLEX parser, 23 million person occurrences and 28 million Location occurrences by the Stanford Named Entity Recognizer[2].

Academic Personal Homepage Domain This dataset is composed of pages about academic people in computer science. We retrieve the name list containing 724,817 authors from DBLP Bibliography website. Using each name as query, we use Google to retrieve the top 3 related pages, and collect a 9GB data corpus containing 2 million webpages. Three basic data types are extracted: 61 million person occurrences, 20 million organization occurrences and 1 million email occurrences.

All experiments are carried on a PC with 2.4GHz Intel Core 2Duo CPU, 1T disk and 3Gb of RAM. We leverage and extend Lucene index to support our designed index structures.

4.6.1 Case Study

This section studies the expressiveness of the CQL language. We study two example tasks (TES and WIE mentioned in Section 4.1) over the aforementioned two domains, and show that these content query tasks could be well supported by CQL in our framework.

Wikipedia Text Domain

On the Wikipedia text domain dataset, we conduct experiments for the Typed-Entity Search (TES) application, to support searching specific types of information based on user inputs. More specifically, we design three tasks, described as follows:

1. Based on #number, retrieve population of a given country.
2. Based on #location, retrieve capital of a given country.

[2]http://nlp.stanford.edu/software/CRF-NER.shtml

	Population	Capital	CEO
Precision	65.5%	90.0%	67.0%

	Professor-Email	Professor-University
Precision	86.0%	96.0%

Figure 4.8: Precision measurement.

3. Based on #person, retrieve CEO of a given company.

Due to space limitation, we only show the CEO search example in query **Q5**:

SELECT #person.val

FROM #person

WHERE (*pattern*("[{CEO of IBM} #person]⟨6⟩")[0.9]

OR *pattern*("[{CEO ?⟨0, 3⟩ #person} IBM]⟨20⟩")[0.5])

AND ∼*prefer*(*TF*("IBM")>10, 0.6, 0.2)

GROUP BY #person

ORDER BY *conf*()

In query **Q5**, the "IBM" keyword is the user input. It can be substituted by any other company names (*e.g.*, Microsoft, Bank of America, *etc.*). Functions such as ∼*prefer*, *TF* are supported in our system. *TF*("IBM") will return the number of "IBM" occurrences in a document . ∼*prefer*(*TF*("IBM")>10, 0.6, 0.2) returns score 0.6 when the value of *TF*("IBM") is greater than 10, or 0.2 otherwise. The query contains two patterns, and the former is more restrictive than the latter. We favor the first pattern by assigning it with a higher weight of 0.9. **Q5** uses the implicit scoring function.

To measure the precision of the three tasks, we use 200 country names as input for population and capital search, and 100 IT companies [3] for CEO search. For each query, we manually check whether top 3 results include the correct answer. The precision results are listed in Figure 4.8. We find that Capital search retrieves the highest precision, while CEO search and Population search are comparatively lower in precision. The low precision is mainly due to the lack of redundancy in Wikipedia corpus where a lot of data type instances usually only appear once. Users can further define more restrictive rules to retrieve more precise results. The result shows that the system can support various TES tasks by returning satisfactory results.

[3] http://www.netvalley.com/top100am_vendors.html

Academic Personal Homepage Domain

On the second dataset of the academic personal homepage domain, we conduct experiments for the Web-Information Extraction (WIE) application, which aims at collecting facts from the Web. This dataset contains a lot of author homepages, upon which we study two tasks:

- Professor-Email extraction. Extract professor-email pairs from the dataset. #professor is defined based on #person.
- Professor-University extraction. #university is defined based on #organization.

Take #professor as example, we define #professor by query **Q6**:

DEFINE DATATYPE #professor.val **AS** #person

WHERE *pattern*("[(prof|professor) #person]⟨4⟩")[0.8]

Query **Q6** defines #professor to be #person with keywords "professor" or "prof" around. With #professor defined, the professor-email pairs could be extracted by query **Q7**:

SELECT #professor.val, #email.val

FROM #professor, #email

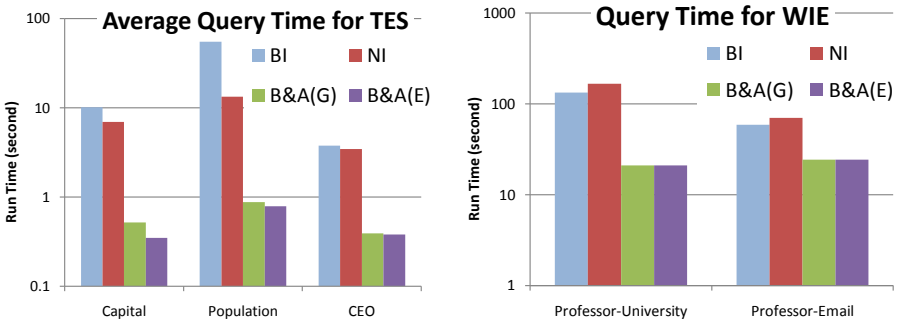
WHERE *pattern*("{#professor ?⟨0,20⟩ #email}")

GROUP BY #professor, #email

ORDER BY *conf*()

Query **Q7** collects professor-email pairs in which #professor and #email appear within window of 20 words. The collected results are ordered by the overall pair frequency (implicitly involved in the scoring calculation of the GROUP BY clause) in the data corpus. This statement extracts 12,174 professor-email pairs and 34,982 professor-university pairs from the data corpus, and we randomly sample 100 pairs from each of them for precision measurement, shown in Figure 4.8. The high precision of results validates the applicability of utilizing CQL language for content query tasks.

In summary, all the queries studied in the above two domains involve all the essential CQL characteristics. We show that various content query tasks could be well supported by CQL. We also find that a content query task usually involves multiple patterns and complex scoring functions which need careful tuning for good performance. DoCQS simplifies this process in allowing administrators to avoid modifying underlying system programs by quickly trying out different queries using CQL.



(a) Wikipedia domain.

(b) Academic domain.

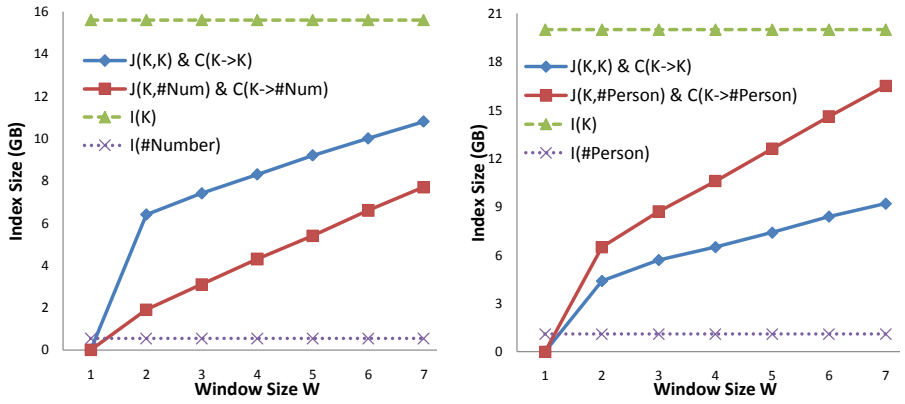
Figure 4.9: Query time comparison.

4.6.2 Time Cost Analysis

For a data-oriented content query system to support arbitrary user-defined queries, performance is a major concern. This section compares the query performance of the tasks defined above using 4 different index selection strategies: 1) BI: using the basic inverted list only; 2) NI: using the neighborhood index (the core index structure of the BE engine described in [14], whose main idea is to store the immediate neighbors of each word in the metadata to speed up phrase queries); 3) B&A(G): using indexes selected by our greedy query plan generation algorithm over basic inverted list and advanced inverted index; 4) B&A(E): using the optimal index plan based on exhaustive plan generation over basic inverted list and advanced inverted index. Using the former two as baseline, we quantitatively demonstrate that the advanced index layer helps greatly improve the query performance. By comparing the latter two cases, we show that our index selection algorithm achieves close to optimal query processing time.

On Wikipedia Text Domain, the average query time on three index structures are compared in Figure 4.9(a). It shows that the advanced index layer achieves six to ten times improvement compared with merely using the basic inverted index, and two to seven times faster than the neighborhood index. That's because the neighborhood index only store immediate neighbors, providing limited optimization room for complex CQL queries. In some cases where few words appear consecutively, the neighborhood index degenerates to the standard inverted index. With the help of the advanced index layer, the average query time of our system is less than one second per query. Results also validate that our greedy algorithm can derive approximately optimal query plans.

On Academic Personal Homepage Domain dataset, we directly measure the execution time for two tasks, shown in Figure 4.9(b). It still shows that our indexing framework works much more efficiently than the other two baseline indices. As there are no consecutive patterns in the query, the performance on the neighborhood index is even worse than on the basic inverted list layer. For these two queries our greedy



(a) Wikipedia domain. (b) Academic domain.

Figure 4.10: Space cost with different window size W.

Domain	Basic Inverted List		Advanced Inverted Index	
	$I(K)$	$I(T)$	$J(K,K)$& $C(K \to K)$	$J(K,T)$& $C(K \to T)$
Wikipedia	15.6GB	1.25GB	9.2GB	12.1GB
Academic	20GB	2GB	7.4GB	19.7GB

Figure 4.11: Space cost in actual implementation.

algorithm can derive the exactly same optimal query plan with the exhaustive algorithm, thus the query time for the latter two cases are the same. Compared with query on the first domain for TES application, it takes longer query time for WIE application because there is no specific words(like country name or company name which appears much less in the data corpus) involved in WIE query. However, since WIE applications are usually carried on offline, the time cost is still acceptable.

4.6.3 Space Cost Analysis

The size of the advanced inverted index is variable by the window size W for the joint index and the contextual index. In Figure 4.10, we show the space cost of the indexes with different window sizes, and also compare them with the space cost of the basic inverted list for keywords and data types. It can be found that, as the window size increases, the space cost increases smoothly, allowing larger window size to support more flexible patterns. In actual implementation, we choose 5 as the default window size. The detailed space cost is summarized in Figure 4.11. As we can see, the space cost is acceptable compared with the original corpus size.

Chapter 5

Query By and For Entities: Relational Entity Search

5.1 Introduction

Traditional entity search methods [19, 82, 5], which return entities that match the relation described by user-given keywords, would fail if the relation is represented by different keywords in Web pages. For example, given a query "microsoft founded by #person," which looks for a person-typed entity who is the *founder of* "microsoft," the entity search engine should return "bill gates" and "paul allen" as the answers. However, if the relation is represented by different keywords in the Web documents (*e.g.*, "started" instead of "founded by"), such a search scheme would not work well.

To address the issue, we propose to study the problem of *relational entity search* in this chapter. In relational entity search, with respect to a desired relation (*e.g.*, *FounderOf*()), given a query (*e.g.*, "microsoft"), the system will rank entities by an automatically learned *relation-specific* ranking function such that a higher-ranked entity (*e.g.*, "bill gates" and "paul allen" as #person) is expected to better match the desired relation for the query. Such a search scheme has two advantages: first, by relying on a relation-specific ranking function, it relieves users from the burden of specifying possible keywords for the target relation; second, we can expect that ranking functions carefully designed for different relations (*e.g.*, *FounderOf*() and *PublisherOf*()) would achieve better performance than a general one.

In order to rank entities for a given query with respect to a desired relation, the ranking function will draw upon the aggregation of their *snippets* – the textual fragments recording how the query and the entity co-occur. Figure 5.1 shows an example of relational entity search, which aims at finding the founders of "microsoft." Formally, given "microsoft" as a query q, we are looking for a person-typed entity e, *i.e.*, $e \in$ #person, such that the relation *FounderOf*("microsoft", e) is true (*i.e.*, "bill gates" and "paul allen" underlined in Figure 5.1). To fulfill such a goal, the relational entity search engine retrieves a list of candidate person entities that ever co-occur with the query "microsoft," and each entity is represented as a bag of snippets describing how entity e and query q are related. Based on such abundance of snippets, the relational entity search engine will draw upon a ranking function to rank the entities.

Entity	Snippets with "microsoft"
e_1: "bill gates"	s_{11}: **Microsoft** was founded by **Bill Gates** and …
	s_{12}: **Bill Gates** met **Microsoft** CEO at his home …
	s_{13}: **Bill Gates** dropped out of college and started **Microsoft.**
e_2: "steve ballmer"	s_{21}: **Steven A. Ballmer** is CEO of **Microsoft.**
	s_{22}: **Steven Ballmer** has headed several **Microsoft** divisions.
e_3: "paul allen"	s_{31}: **Paul Allen** is best known to be the co-founder of **Microsoft**.
	s_{32}: **Microsoft** is founded by … **Paul Allen** in 1970's …
e_4: "jerry yang"	s_{41}: **Microsoft** CEO met Yahoo co-founder **Jerry Yang**…
	s_{42}: If **Jerry Yang** is upset that **Microsoft** bid to buy the company …

Figure 5.1: Example entities and instances (snippets).

Hence, to realize effective relational entity search, we must essentially exploit the redundancy of the Web – an insight that many Web applications (*e.g.*, Web-based QA [20, 11], Web IE [22]) have exploited. In large corpora like the Web, true facts are often repeated many times, and we can leverage such *redundancy* of Web data to achieve a more reliable entity ranking. Taking Figure 1 as an example, different snippets (s_{11}, s_{12}) mention the fact that "bill gates" founded "microsoft," based on which we can confidently conclude that "bill gates" is a correct answer for *FounderOf*("microsoft", e).

However, while redundancy of the Web is an opportunity to exploit, it comes with inherent noise which hampers effective ranking in our setting. Such *noisy redundancy* exists as not all the snippets represent information relevant to the target relation (*e.g.*, s_{12} to the relation *FounderOf*()). Failing to filter out such noisy snippets will tempt the ranking model to recognize a negative sense of evidence for supporting the desired relations, *e.g.*, treating s_{12} as an evidence for *FounderOf*(). Existing redundancy-based Web applications [19, 82, 5] rely on manually-crafted or user-given keywords (*e.g.*, "founded by") to match relevant snippets, whereas, in relational entity search, without such keywords as given, the problem boils down to learning a relation-specific ranking function that can effectively filter noisy snippets.

Learning such a noise-robust relational entity ranking function is non-trivial. In order to filter out noisy snippets, one straightforward solution is to first annotate each (q, e, s) triplet (*e.g.*, by annotating s_{11} and s_{13} as positive, s_{12} as negative), and train a classifier for each relation. Unfortunately, such a solution is impractical as even a single entity is associated with a large number of snippets; although one can sample a small set of snippets for annotation, it is difficult to guarantee that the sampled snippets would cover different representations of the relation to achieve good generalization capability.

In this chapter, rather than requiring detailed snippet-level annotations, we propose to utilize entity-level annotations as *distant supervision* [53] for learning the ranking model. Such entity-level annotations are usually easy to obtain from various public knowledge bases such as Freebase and Wikipedia. For example, in Figure 5.1, according to Freebase, we can label "bill gates" and "paul allen" as positive, without indicating

which snippets actually match the desired relation (s_{11} and s_{13} do, but not s_{12}), and we are mandated to learn the ranking function for *FounderOf*() based on such coarse-grained annotations.

Therefore, we abstract our task as the *distantly supervised ranking* problem which aims at learning a relation-specific ranking function for entity search based on distant supervision. In particular, in the *offline training phase*, with respect to a target relation, given a set of training queries with labeled entities and unlabeled snippets, one needs to derive a ranking function, such that, in the *online searching phase*, given a new query, the ranking function could correctly identify its positive entities that satisfy the target relation based on their associated snippets. Towards addressing the distantly supervised ranking problem, we are facing two challenges:

First, in terms of *accuracy*, due to the lack of detailed snippet labels, we have to filter noise before leveraging the redundancy for ranking. Taking Figure 5.1 as example, the ranking function should recognize s_{12} as a noisy snippet, and only rely on the snippets of s_{11} and s_{13} to rank entity e_1.

Second, to facilitate an online search engine, the ranking function must be *efficient*. In particular, we should resort to existing inverted indexes to fetch features for snippet filtering, and therefore we can only choose features that are "indexable." Moreover, since such index traversing is also time consuming, the number of "indexable" features should also be limited by a small budget.

In order to exploit noisy redundancy accurately and efficiently, we believe *the key lies in leveraging redundancy itself*. As our insight, a positive entity will have more relevant snippets as support, which we identify as the *redundancy ranking principle*. Moreover, there always exist some common keywords/phrases that are indicative of the target relation in a relevant snippet (*e.g.*, "founded by" for *FounderOf*()). We name such keywords/phrases as *indicative patterns* and leverage such *pattern redundancy* to address the two challenges. First, in terms of accuracy, we can identify such indicative patterns by their redundancy for noise filtering. Taking Figure 5.1 as an example, since the pattern "founded by" co-occurs most of time with positive entities "bill gates" and "paul allen," we can confidently conclude that "founded by," rather than "met ... CEO," is a good indicative pattern for *FounderOf*(), and thus s_{11} and s_{32} should be chosen as evidences and s_{12} should be filtered as noise. Second, in terms of efficiency, due to pattern redundancy, a small number of indicative patterns can already cover a large range of relevant snippets and retrieve almost all positive entities (by covering at least one of their relevant snippets) for ranking, and thus we can limit the number of patterns to be small to fulfill the efficiency requirement.

Based on such an insight, we develop the *Pattern-based Filter Network*, named *PFNet*, in the framework of Markov network, as our solution. Supported by the power of *probabilistic models*, we encode our redundancy ranking principle in a principled way: to tackle with noise and improve efficiency, PFNet selects a restricted

set of indicative patterns to fetch a small size of snippets as evidences for the candidate entities by inverted indexes, and filter other snippets as noise; the candidate entities are ranked based on the aggregation of such chosen evidences. A step-wise greedy algorithm is proposed to estimate PFNet efficiently.

To validate the effectiveness of the proposed PFNet model, we performed large-scale evaluation over 100 million web pages from the ClueWeb09 dataset (on a cluster of 30 nodes), for a variety of six relational entity search tasks (e.g., *FounderOf*() and *PlaceOfBirth*()). In order to demonstrate the necessity of automatically learning relational ranking functions and the importance of leveraging redundancy and filtering noise, we compared our algorithm with different baseline methods including EntityRank which is an unsupervised ranking model proposed in [19], multi-instance learning [63] which models noise but does not leverage redundancy, and SVMRank [35] which directly leverages redundancy without noise filtering. We observed consistent improvement (NDCG@5 +15% in average over all relations) compared with those baselines.

5.2 Related Work

There are several related efforts of relational entity search in the literature. Cheng et al. propose EntityRank, a domain-independent probabilistic entity ranking model [19], where local recognition and global access information are integrated in an unsupervised manner. In Zhou et al.'s DoCQS system [82], the searching capability has been distilled by asking users to manually craft the ranking rules for their inquired relations. Banerjee et al. [5] study to answer quantity consensus queries, where each answer is a tight quantity interval extracted from evidences in thousands of snippets. In addition, our work is also closely related to Web-based QA. Traditional Web-based QA work [43, 11, 2] heavily relies on standard search engines to retrieve relevant documents based on the input question and then extract answers according to some predefined templates, which would fail if the answer is represented by different keywords in the Web page. In contrast, our work leverages the abundance of relational data as training data to automatically learn more robust and accurate retrieval models for specific relation types. Thus, our work could be viewed as an important vehicle to support traditional Web-based QA.

The idea of leveraging Web redundancy is first proposed by Clarke et al. [20], and it is further validated and widely adopted in various Web applications such as Web-based QA [11, 51], Web IE [22] and entity search [19]. Most of such works rely on manually-crafted patterns or user-given queries to retrieve relevant snippets, and they model redundancy by different approaches, *e.g.*, [51, 22] measure the redundancy of the query-answer co-occurrences by Pointwise Mutual Information (PMI); while in [19], Cheng et al. use a global access layer to aggregate the redundancy of snippet scores. In contrast, without patterns that are indicative

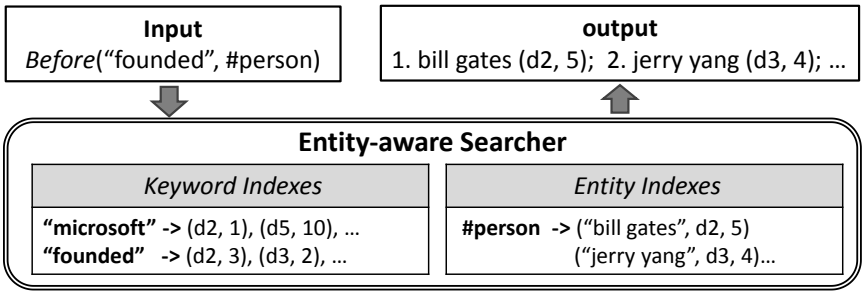


Figure 5.2: Entity-aware searcher.

of the target relation as given, starting with snippets where the query and the entity simply co-occur, we filter noisy snippets by automatically learned patterns, and rank entities based on the filtered snippets according to the redundancy ranking principle.

Besides, some recent work has explored the idea of distant supervision. Mintz et al. utilize Freebase data as distant supervision to learn relational classifiers [53]. In their work, various types of features ranging from lexical (N-gram and POS tagging patterns) and syntactic (dependency parsing) features are introduced to enhance the classification performance. However, they do not consider the potential noise in the training data, and use all the snippets to train their classifier. Riedel et al. realize the existence of noise and propose to utilize the distant supervision in a more sophisticated manner [63]: they assume that positive entities should have *at least one* evidence snippet and solve the problem by Multi-Instance Learning [4]. Hoffmann et al. extend it by presenting a conditional extraction model to learn relations from heuristically labeled training data and combat with noise [30]. Although previous studies have noticed the adverse impact introduced by the noisy unlabeled data, their solutions are not directly applicable to our relational entity search problem. First, in terms of efficiency, they heavily depend on the expensive IE features, *e.g.*, POS tagging and dependency parsing, to distinguish noisy snippets, which are infeasible for an online system; second, in terms of accuracy, the "redundancy" is not explicitly modeled in their methods, that would make the model vulnerable to erroneous positive snippets in negative entities. In our work, we restrict our model to the "indexable" features for online processing purpose and exploit the redundancy within the associated snippets to solve the relational entity search problem.

5.3 Exploring Noisy Redundancy for Relational Entity Search

In this section, we describe our general framework for relational entity search. As the major component of the framework, the *distantly supervised ranking* problem is formally defined in the end of the section.

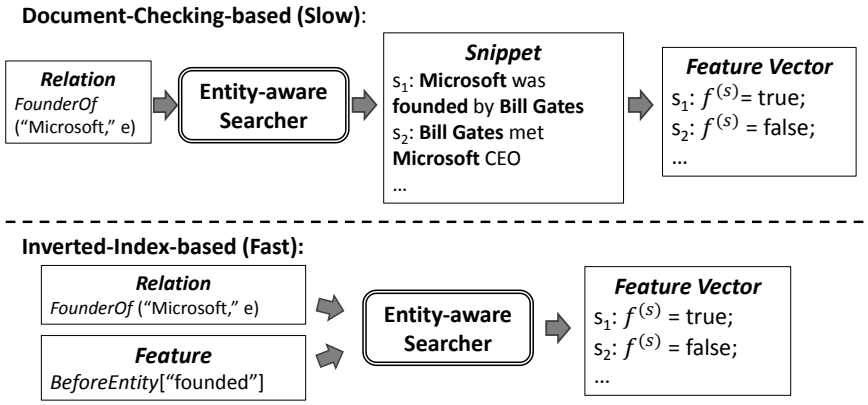


Figure 5.3: Two different feature fetching methods to obtain $f^{(s)} = BeforeEntity$["founded"].

5.3.1 Relational Entity Search Framework

In relational entity search, given a target relation $r(q, \#\mathsf{E})$, in which we are interested in the entity e of type $\#\mathsf{E}$ that forms relation r with q. For example, for query *FounderOf*("microsoft", #person), we need to find $e \in \#\mathsf{person}$ (*e.g.*, "bill gates" and "paul allen") such that *FounderOf*("microsoft", e) is true. The desired output is a list of entities $e \in \#\mathsf{E}$, ranked by the confidence that r holds between q and e.

To retrieve the candidate entities e, we appeal to an *entity-aware searcher* (*e.g.*, [17, 19]), which, essentially, is a document retrieval system with an additional index of entities to support entity finding. Figure 5.2 shows an example index for entity type #person, where each *posting* records a specific entity (*e.g.*, "bill gates") and its occurrence in the corpus (*e.g.*, 5th word in document $d2$). With entity position recorded in the inverted index, the searcher can support a set of proximity checking operations: for example, for the input of *Before*["founded", #person], the searcher will return all the #person entities which appear before the keyword "founded" in some snippets.

Based on such an entity-aware searcher, we design our relational entity search framework in Figure 5.4: a set of relation-specific ranking models is learned in the offline training phase in advance; and in the online searching phase, such learned models would be chosen to answer the input query according to the target relation.

In the *offline training phase*, given a set of known relations, we utilize the entity-aware searcher to obtain candidate entities and construct a training corpus. Taking the relation *FounderOf*("microsoft", {"bill gates", "paul allen"}) as an example, we will use the entity-aware searcher to find a list of #person entities (*e.g.*, "bill gates") that ever co-occur with the term "microsoft," and associate each candidate entity with a set of snippets. Then, "bill gates" and "paul allen" will be labeled as positive, and other entities as negative, leaving all the snippets unlabeled.

With only such coarse entity-level annotations, we are mandated to learn a relation-specific ranking

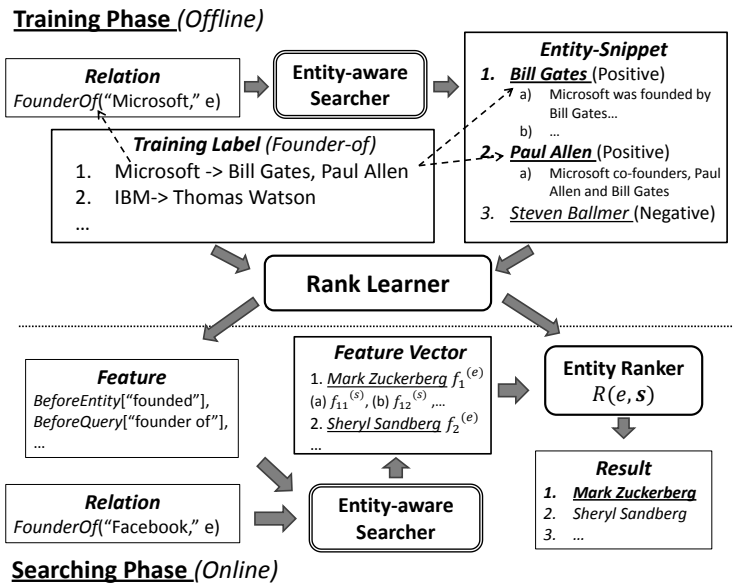


Figure 5.4: Relational entity search framework.

function $R(e, \mathbf{s})$, which takes an entity e and its supporting snippets $\mathbf{s}$ as input, and produces a ranking score to rank the candidate entities according to a set of features defined over e and $\mathbf{s}$ with respect to the target relation r. We name such a learning problem as the *distantly supervised ranking* problem. As the name indicates, first, we need to explore the indirect supervision defined on entities; second, as a practical ranking problem, the ranking function R must be *executable* online. Before we give the formal definition and solution of the *distantly supervised ranking* problem, we would first discuss the properties a practical ranking function R should have for efficient online execution purpose.

The *online searching phase* will execute the learned ranking model against new queries. As shown in Figure 5.4, given a query, *e.g.*, *FounderOf*("facebook", e), the entity-aware searcher will look for the related #person entities and snippets, then a relation-specific entity ranker will predict the ranking among the retrieved entities.

As an online system, efficiency is the main concern: the major bottleneck arises in fetching the ranking features. Assuming that we are interested in a snippet feature indicating whether the keyword "founded" appears before the entity, denoted as *BeforeEntity*["founded"], there are two different approaches to obtain such a feature, as shown in Figure 5.3. The first approach, based on document checking, needs to fetch all snippets first, and then extract features from the detailed snippet contents. Although straightforward, it is quite inefficient as it requires a large number of I/O operations, especially when the number of snippets is large. In this work, we would adopt the second approach which takes advantage of inverted indexes: with all positions of keywords and entities recorded in an inverted index, *BeforeEntity*["founded"] can be calculated for all snippets by joining the "founded" index and the #person index.

Entity Features $f_k^{(e)}$	
idf	Inverse document frequency of entity in corpus
tf	The number of document co-occurred with query
len	Number of words the entity contains
Snippet Features $f_k^{(s)}$	
dis	Word distance between query and entity in snippet
qtf	Query frequency in the document of snippet
etf	Entity frequency in the document of snippet
pos	Snippet position in the document
BeforeQuery[w]	If w appears before query within 3-word window
AfterQuery[w]	If w appears after query within 3-word window
BeforeEntity[w]	If w appears before entity within 3-word window
AfterEntity[w]	If w appears after entity within 3-word window
Around[w]	If w appears around query-entity within 20-word

Figure 5.5: Features for relational entity search.

To adapt to the inverted index-based approach, the ranking model can only use "indexable" features, or more specifically, the ones that are supported by the entity-aware searcher. Figure 5.5 illustrates all the features we designed for our framework. In general, the features can be classified into two types: entity features $f_k^{(e)}$, *e.g.*, $tf(e)$, and snippet features $f_k^{(s)}$, *e.g.*, *BeforeEntity*["founded"](s). The ranking function R will rely on such features to estimate the relevance of an entity to the given query. For example, *BeforeEntity*["founded"](s_{11})=1, which means "founded" occurs before "bill gates" in s_{11}, indicates that s_{11} may serve as an evidence supporting the relation *FounderOf*("microsoft", "bill gates"). The value of entity features can be obtained when we are jointing the query index with the entity index as shown in Figure 5.2; while to get the snippet features, we need to perform additional join with the selected entities as shown in Figure 5.3, which would be the major bottleneck of efficiency.

To achieve fast online retrieval, we need to limit the number of snippet features, which requires additional index checking. We set M to be the maximal number of the selected snippet features. As a result, the designed snippet features should yield relatively high recall to cover more snippets. Such requirement prohibits us from adopting the high-precision low-recall features in traditional information extraction work (*e.g.*, [51] used all words between the query and the entity as one feature, like "which was recently founded by"). Conversely, we limit each w to be unigram (*e.g.*, "founder") or bigram ("founded by") in our feature design.

The relation-specific ranking model is the focus of this chapter. In the next section, we will formally study the problem of how to effectively learn such models with indirect supervision, *i.e.*, the *distantly supervised ranking* problem.

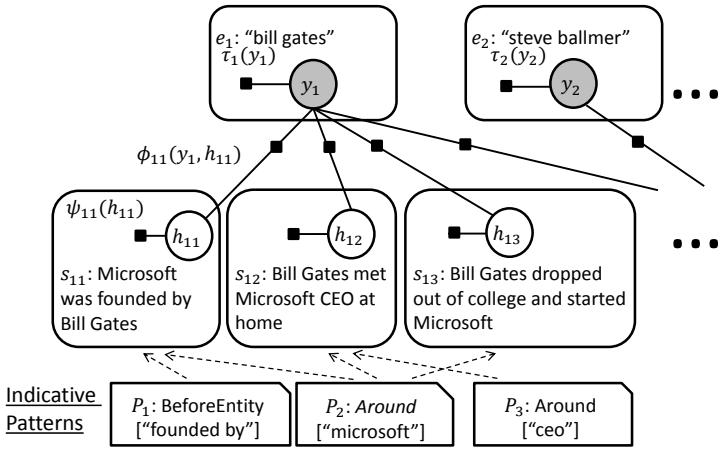


Figure 5.6: Pattern-based filter network.

5.3.2 Distantly Supervised Ranking

In this section, we would formally define the *distantly supervised ranking* problem.

With respect to a target relation r, *e.g.*, *FounderOf*(), we are given a set of training queries. For each query q, *e.g.*, "microsoft" (to simplify the description in the following discussion, we will only use this query for illustration purpose), we obtain a list of entities $\mathbf{e} = \{e_1, ..., e_{|\mathbf{e}|}\}$ (*e.g.*, "bill gates" and "paul allen") returned by the entity-aware searcher; each entity e_i is associated with a set of snippets $\mathbf{s_i} = \{s_{i1}, ..., s_{i|\mathbf{s_i}|}\}$. s_{ij} is defined as a fixed-length text fragment where entity e_i and query q co-occur. We assign a label y_i to each $e_i \in \mathbf{e}$: if the relation $r(e_i, q)$ is true, e_i is labeled as positive ($y_i = 1$); otherwise negative ($y_i = 0$). Entity e_i and snippet s_{ij} are characterized by a set of ranking features $f_k^{(e)}$ and $f_k^{(s)}$ defined in Figure 5.5.

For efficiency concern, we have to limit the number of snippet features that require additional index checking. Denoting $\mathcal{F}_{\text{all}} = \{f_1^{(s)}, ..., f_{|\mathcal{F}_{\text{all}}|}^{(s)}\}$ as the whole candidate snippet feature set, the final model can only choose M snippet features $\mathcal{F} \subseteq \mathcal{F}_{\text{all}}$.

As our objective, we aim to learn a ranking function $R(e_i, \mathbf{s_i})$ for the target relation r, such that the candidate entities are ordered by the confidence of being in the true relation. Using the language of probability, $R(e_i, \mathbf{s_i})$ can be described as $P(y_i = 1|e_i, \mathbf{s_i})$, the probability of e_i being true given entity features $f_k^{(e)}$, and snippet features $f_k^{(s)} \in \mathcal{F}$. As a result, the goal of distantly supervised ranking is to find the optimal ranking function in the form of $P(y_i = 1|e_i, \mathbf{s_i})$, which can correctly rank the annotated entities for the target relation, based on a limited size of snippet features satisfying $|\mathcal{F}| \leq M$.

5.4 Pattern-Based Filter Network

There are two challenges embedded in the distantly supervised ranking problem: one is *accuracy*, namely we need to filter noise inherited in the coarse entity-level annotations so as to estimate an accurate ranking function $R(e_i, \mathbf{s_i})$; another is *efficiency*, that is we can only employ a small amount of "indexable" features when solving the problem.

As discussed in Section 5.1, to address the two challenges, we would rely on the *redundancy ranking principle* – intuitively, a positive entity will have more relevant snippets as evidences; furthermore, as different relevant snippets will have different contributions (*e.g.*, a snippet with small *dis* represents strong relatedness between the query and entity, and thus tends to be more important), we hypothesize that the positiveness of an entity should depend on the summation of the contribution, rather than the number, of its relevant snippets. If we define $\mathbf{h_i} = \{h_{i1}, ..., h_{i|\mathbf{h_i}|}\}$ where $h_{ij} \in \{0, 1\}$ denotes whether s_{ij} is an *evidence snippet* and a_{ij} measures the contribution of s_{ij} to e_i, the redundancy ranking principle could be formalized as follows,

Definition 3 (Redundancy Ranking Principle): Given entities e_i and e_j with the same entity features, we have

$$\begin{aligned} & \sum_{s_{ij} \in \mathbf{s_i} \wedge h_{ij}=1} a_{ij} > \sum_{s_{lj} \in \mathbf{s_j} \wedge h_{lj}=1} a_{lj} \\ \rightarrow \quad & P(y_i = 1 | \mathbf{h_i}, e_i, \mathbf{s_i}) > P(y_l = 1 | \mathbf{h_l}, e_l, \mathbf{s_l}) \end{aligned} \tag{5.1}$$

where a_{ij} is defined as the weighted summation of the chosen snippet features of s_{ij},

$$a_{ij} = \sum_{f_k^{(s)} \in \mathcal{F}} w_k^{(s)} f_k^{(s)}(s_{ij}) \tag{5.2}$$

where $w_k^{(s)}$ is the importance weight of snippet features. ∎

As the prerequisite to apply the redundancy ranking principle, one needs to determine which snippets should serve as evidences, *i.e.*, $h_{ij} = 1$. As proposed in Section 5.1, we utilize *indicative patterns*, some common patterns that are indicative of the target relation r (*e.g.*, *FounderOf*()) between the entity e and query q, to identify the evidence snippets. Formally, we define $\mathbf{P} = \{\mathcal{P}_1, ..., \mathcal{P}_{|\mathbf{P}|}\}$ as the indicative pattern set, where each indicative pattern is a binary function $\mathcal{P}_k(s_{ij}) \in \{0, 1\}$ defined over the chosen snippet features $\mathcal{F}$, *e.g.*, $\mathcal{P}_k(s_{ij}) :=$ *BeforeEntity*["founded by"] $\wedge$ *dis* < 5. Assuming such an indicative pattern set $\mathbf{P}$ is given (we will discuss how to learn $\mathbf{P}$ in the later section), we can predict if a snippet s_{ij} is a positive

evidence by matching it against $\mathbf{P}$ as follows,

Definition 4 (Evidence Snippet): For a given set of indicative patterns $\mathbf{P}$, snippet s_{ij} is an *evidence snippet* (*i.e.*, $h_{ij} = 1$) if and only if $\exists\ \mathcal{P}_k \in \mathbf{P}, \mathcal{P}_k(s_{ij}) = 1$. We use $\mathbf{s_i}^{(\mathbf{P})} := \{s_{ij} \in \mathbf{s_i} | \exists \mathcal{P}_k \in \mathbf{P}, \mathcal{P}_k(s_{ij}) = 1\} \subseteq \mathbf{s_i}$ to denote the evidence snippet set for e_i. ∎

As discussed in Section 5.1, we can address the two challenges based on the redundancy ranking principle defined upon the notion of evidence snippet: as such indicative patterns commonly exist in relevant snippets across different positive entities, noisy snippets can be filtered by the redundancy of indicative patterns; in addition, the diversity of such patterns is usually small, which renders us the opportunity to limit the number of patterns without hurting the accuracy.

As a result, there are two objectives clearly stated in the redundancy ranking principle to address the two challenges: first, filtering noisy snippets, and second, scoring the positive snippets based on chosen evidences. To fulfill this modeling assumption, we propose *Pattern-based Filter Network* (PFNet), which consists of $|\mathbf{e}|$ tree-structured Markov networks [39], as shown in Figure 5.6. The proposed PFNet forms a two-layer tree structure and each layer corresponds to one particular modeling objective:

- **Noise Filtering**: At the leaf layer (bottom), PFNet selects a set of indicative patterns $\mathbf{P}$ to build up factor $\psi_{ij}(h_{ij})$ for effectively distinguishing noisy snippets, *i.e.*, $P(\mathbf{h_i}|\mathbf{s_i})$;
- **Evidence Aggregation**: At the root layer, to predict y_i, *i.e.*, the conditional probability of $P(y_i|\mathbf{h_i}, e_i, \mathbf{s_i})$, PFNet uses $\tau_i(y_i)$ to model the features from entity e_i itself, and $\phi_{ij}(y_i, h_{ij})$ for modeling the redundancy within snippets. In the aggregation of snippet redundancy, $\phi_{ij}(y_i, h_{ij})$ only aggregate the contribution a_{ij} of chosen evidence snippets s_{ij} with $h_{ij} = 1$.

As a result, each tree in PFNet specifies the conditional probability of $P(y_i, \mathbf{h_i}|e_i, \mathbf{s_i})$ for an entity $e_i \in \mathbf{e}$ being positive (*e.g.*, those underlined e_i in Figure 5.1) and the snippet s_{ij} being an evidence snippet by the factors of $\psi_{ij}(h_{ij})$, $\phi_{ij}(y_i, h_{ij})$ and $\tau_i(y_i)$, *i.e.*,

$$\begin{aligned} P(y_i, \mathbf{h_i}|e_i, \mathbf{s_i}) &= P(y_i|\mathbf{h_i}, e_i, \mathbf{s_i}) P(\mathbf{h_i}|\mathbf{s_i}) \\ &\propto \tau_i(y_i) \prod_{j=1}^{|\mathbf{s_i}|} \phi_{ij}(y_i, h_{ij}) \psi_{ij}(h_{ij}) \end{aligned} \tag{5.3}$$

In the following discussion, we will illustrate the design of each layer in PFNet in detail.

5.4.1 Noise Filtering

As required by the redundancy ranking principle, to avoid distraction from noisy snippets, we use indicative patterns to identify evidence snippets, encoded in $\psi_{ij}(h_{ij})$. In a standard Markov network, factors are defined via exponential functions. However, due to the positiveness of an exponential function, all snippets, including the noisy ones, will have non-zero probabilities of being an evidence, especially when the volume of noisy snippets is larger than the evidence snippets. This will bias the prediction of y_i for entity e_i. Moreover, this setting can hardly scale up to a large data set: one has to repeatedly infer h_{ij} when estimating the model, which makes the training process prohibitively slow. As a result, we adopt a "hard filter" design for $\psi_{ij}(h_{ij})$ as,

$$\psi_{ij}(h_{ij}) = \begin{cases} 1 & \text{if } s_{ij} \in \mathbf{s_i}^{(\mathbf{P})} \wedge h_{ij} = 1 \text{ or} \\ & \quad s_{ij} \notin \mathbf{s_i}^{(\mathbf{P})} \wedge h_{ij} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

where $\psi_{ij}(h_{ij})$ is modeled as a binary factor to emphasize the effect of matching against the indicative pattern set $\mathbf{P}$.

5.4.2 Evidence Aggregation

The root layer of PFNet models the prediction of y_i, which indicates if e_i is a positive entity (thus an answer to the query). Specifically, we want to specify the interaction between y_i and the latent variables h_{ij} and observations (e_i and s_{ij}) by factors $\tau_i(y_i)$ and $\phi_{ij}(y_i, h_{ij})$.

Factor $\tau_i(y_i)$ is designed to capture the contribution from the general entity features on e_i. One property that $\tau_i(y_i)$ should follow is that when we failed to find any evidence for e_i, it is more likely to be a negative entity. Given a set of indicative patterns $\mathbf{P}$, denote $\mathbf{e}^{(\mathbf{P})}$ as the set of *covered entities*, where each entity should have at least one evidence snippet. Formally, we define $\mathbf{e}^{(\mathbf{P})} := \{e_i \in \mathbf{e} | \mathbf{s_i}^{(\mathbf{P})} \neq \emptyset\}$. If $e_i \notin \mathbf{e}^{(\mathbf{P})}$, we should give $y_i = 1$ a small probability; otherwise, the prediction of y_i depends on its features $f_k^{(e)}$. To achieve this requirement, we define $\tau_i(y_i)$ as,

$$\begin{aligned} \tau_i(1) &= \begin{cases} \varepsilon & \text{if } e_i \notin \mathbf{e}^{(\mathbf{P})} \\ \exp[\sum_k w_k^{(e)} f_k^{(e)}(e_i)] & \text{if } e_i \in \mathbf{e}^{(\mathbf{P})} \end{cases} \\ \tau_i(0) &= 1 \end{aligned} \tag{5.5}$$

where ε is a small positive constant. Note that, we set $\tau_i(0) = 1$, since only the relative value between $\tau_i(1)$

and $\tau_i(0)$ matters in the Markov network.

$\phi_{ij}(y_i, h_{ij})$ explicitly encodes the aggregation of snippet contribution in the redundancy ranking principle. First, for noisy snippets (with $h_{ij} = 0$), we set $\phi_{ij}(0,0) = \phi_{ij}(1,0) = 1$, indicating the existence of noisy snippets should not affect the prediction of y_i. Second, for an evidence snippet (with $h_{ij} = 1$), how s_{ij} affects $P(y_i = 1|\mathbf{h_i}, e_i, \mathbf{s_i})$ depends on its contribution a_{ij}. Similar to Eq. 5.5, we use the exponential function to incorporate a_{ij} in $\phi_{ij}(1,1)$ and set $\phi_{ij}(0,1)$ to be 1 as

$$\phi_{ij}(y_i, h_{ij}) = \begin{cases} \exp(a_{ij}) & \text{if } y_i = 1 \text{ and } h_{ij} = 1 \\ 1 & \text{otherwise;} \end{cases} \tag{5.6}$$

We can observe that Eq. 5.6 strictly reflects the redundancy ranking principle, *i.e.*, the resulting $P(y_i = 1|\mathbf{h_i}, e_i, \mathbf{s_i})$ is a function monotonically increase with $\sum_{s_{ij} \in \mathbf{s_i} \wedge h_{ij}=1} a_{ij}$. By definition in Eq. 5.3, we have

$$\begin{aligned} & P(y_i = 1|\mathbf{h_i}, e_i, \mathbf{s_i}) \\ = & \frac{\tau_i(1) \prod_j \phi_{ij}(1, h_{ij})}{\tau_i(1) \prod_j \phi_{ij}(1, h_{ij}) + \tau_i(0) \prod_j \phi_{ij}(0, h_{ij})} \\ = & \frac{\tau_i(1) \exp(\sum_{s_{ij} \in \mathbf{s_i} \wedge h_{ij}=1} a_{ij})}{1 + \tau_i(1) \exp(\sum_{s_{ij} \in \mathbf{s_i} \wedge h_{ij}=1} a_{ij})} \end{aligned} \tag{5.7}$$

which clearly satisfies the monotonic property.

5.4.3 Objective of PFNet

The graph structure described in Figure 5.6 with factors designed in Eq. 5.4, Eq. 5.5 and Eq. 5.6 together define our PFNet model, which represents the conditional probability $P(y_i, \mathbf{h_i}|e_i, \mathbf{s_i})$, as defined in Eq. 5.3. Due to the lack of detailed snippet-level labels h_{ij} in training data, the only supervision we have comes from $P(y_i|e_i, \mathbf{s_i})$. To obtain the representation of $P(y_i|e_i, \mathbf{s_i})$, we need to marginalize $P(y_i, \mathbf{h_i}|e_i, \mathbf{s_i})$ over all possible configurations of $\mathbf{h_i}$:

$$\begin{aligned} P(y_i|e_i, \mathbf{s_i}) &= \sum_{\mathbf{h_i}} P(y_i, \mathbf{h_i}|e_i, \mathbf{s_i}) \\ &\propto \sum_{\mathbf{h_i}} \tau_i(y_i) \prod_{j=1}^{|\mathbf{s_i}|} \phi_{ij}(y_i, h_{ij}) \psi_{ij}(h_{ij}) \end{aligned} \tag{5.8}$$

Because of the product term in Eq. 5.8, only one specific $\mathbf{h_i}$ configuration leads to a non-zero $P(y_i, \mathbf{h_i}|e_i, \mathbf{s_i})$ (assign 1 only to the evidence snippets and 0 to all the others). Therefore, $P(y_i|e_i, \mathbf{s_i})$ can be simplified as

$$\begin{aligned} P(y_i|e_i, \mathbf{s_i}) &= P(y_i, \mathbf{h_i} = \hat{\mathbf{h}}_\mathbf{i}(\mathbf{P})|e_i, \mathbf{s_i}) \\ &\propto \tau_i(y_i) \prod_{j:s_{ij} \in \mathbf{s_i}^{(\mathbf{P})}} \phi_{ij}(y_i, 1) \end{aligned} \tag{5.9}$$

where $\hat{\mathbf{h}}_\mathbf{i}(\mathbf{P})$ is the configuration of $\mathbf{h_i}$ corresponding to the given $\mathbf{P}$, and for each $\hat{h}_{ij}(\mathbf{P}) \in \hat{\mathbf{h}}_\mathbf{i}(\mathbf{P})$, $\hat{h}_{ij}(\mathbf{P}) = 1$ if $s_{ij} \in \mathbf{s_i}^{(\mathbf{P})}$ and $\hat{h}_{ij}(\mathbf{P}) = 0$ otherwise.

By substituting $\tau_i(y_i)$ and $\phi_{ij}(y_i, h_{ij})$ in Eq. 5.9 with Eq. 5.5 and Eq. 5.6, we obtain the complete formula of $P(y_i = 1|e_i, \mathbf{s_i})$ as,

$$P(y_i = 1|e_i, \mathbf{s_i}) = \begin{cases} \frac{\varepsilon}{1+\varepsilon} & e_i \notin \mathbf{e}^{(\mathbf{P})} \\ \frac{1}{1+e^{-d_i(\vec{w}, \mathcal{F}, \mathbf{P})}} & e_i \in \mathbf{e}^{(\mathbf{P})} \end{cases} \tag{5.10}$$

where $\vec{w}$ is a feature weighting vector including $w_k^{(e)}$ and $w_k^{(s)}$, and $d_i(\vec{w}, \mathcal{F}, \mathbf{P})$ is defined as follows,

$$\begin{aligned} & d_i(\vec{w}, \mathcal{F}, \mathbf{P}) \\ = & \sum_k w_k^{(e)} f_k^{(e)}(e_i) + \sum_{s_{ij} \in \mathbf{s_i}^{(\mathbf{P})}} \sum_{f_k^{(s)} \in \mathcal{F}} w_k^{(s)} f_k^{(s)}(s_{ij}) \end{aligned} \tag{5.11}$$

Eq. 5.10 clearly illustrates the probability of an entity being positive depends on its own features $f^{(e)}$ and the snippet features $f^{(s)}$ from the associated evidence snippets: if e_i is an entity without any evidence snippet, $P(y_i = 1|e_i, \mathbf{s_i})$ equals to a small constant probability $\frac{\varepsilon}{1+\varepsilon}$; otherwise, $P(y_i = 1|e_i, \mathbf{s_i})$ depends on $d_i(\vec{w}, \mathcal{F}, \mathbf{P})$, the weighted summation of features from e_i and its evidence snippets $s_{ij} \in \mathbf{s_i}^{(\mathbf{P})}$, incorporated in a logistic function.

With $P(y_i = 1|e_i, \mathbf{s_i})$ defined in Eq. 5.10, following the maximal likelihood principle, our objective is to find the optimal set of feature set $\mathcal{F}$, indicative pattern set $\mathbf{P}$, and feature weighting $w_k^{(e)}$ and $w_k^{(s)}$, which maximizes the log likelihood $\mathcal{L}(\mathcal{F}, \vec{w}, \mathbf{P}; \mathbf{y})$ over the annotated entities for the target relation. Note that in entity ranking task, the data set is usually heavily unbalanced: given a query, there are a lot more negative entities than positive ones. Therefore, we set a cost parameter $\lambda > 1$ to emphasize the importance of positive entities. Thus, the objective function $\mathcal{L}(\mathcal{F}, \vec{w}, \mathbf{P}; \mathbf{y})$ is given as follows,

$$
\begin{aligned}
& \underset{\mathcal{F},\vec{w},\mathbf{P}}{\operatorname{argmax}} \ \mathcal{L}(\mathcal{F},\vec{w},\mathbf{P};\mathbf{y}) \\
= & \underset{\mathcal{F},\vec{w},\mathbf{P}}{\operatorname{argmax}} \sum_{e_i \notin \mathbf{e}^{(\mathbf{P})}} \lambda y_i \log \frac{\varepsilon}{1+\varepsilon} + (1-y_i)\log\frac{1}{1+\varepsilon} \\
+ & \sum_{e_i \in \mathbf{e}^{(\mathbf{P})}} \lambda y_i \log \frac{1}{1+e^{-d_i(\vec{w},\mathcal{F},\mathbf{P})}} \\
+ & (1-y_i)\log\frac{e^{-d_i(\vec{w},\mathcal{F},\mathbf{P})}}{1+e^{-d_i(\vec{w},\mathcal{F},\mathbf{P})}}
\end{aligned}
$$

subject to

$$
|\mathcal{F}| \leq M \tag{5.12}
$$

5.4.4 Model Learning

As Eq. 5.12 indicates, the goal of PFNet model learnging boils down to searching for the optimal $\mathcal{F}$, $\vec{w}$ and $\mathbf{P}$, which maximize $\mathcal{L}(\mathcal{F},\vec{w},\mathbf{P};\mathbf{y})$. Unfortunately, such an optimization problem is generally difficult. First, it is an NP-complete problem: one needs to solve $\binom{|\mathcal{F}_{\text{all}}|}{M}$ nonlinear subproblems to find the optimal solution. (The problem can be reduced to a subset selection problem of selecting $\mathcal{F}$ from $\mathcal{F}_{\text{all}}$ with given $\vec{w}$ and $\mathbf{P}$, which is known to be NP-complete.) Second, even for a single query, it might contain millions of snippets. The learning procedure must be efficient to be applied onto such a large-scale environment.

To derive a scalable learning algorithm, we start with a simplification of the objective function. As defined in Section 5.4.1, an indicative pattern could be any boolean function containing arbitrary number of predicates (*e.g.*, $\mathcal{P}_k(s_{ij}) :=$ *BeforeEntity*["founded by"] $\wedge$ *dis* < 5). However, in our experiments we find that higher order of predicates does not contribute to the performance while complicates the learning process. Therefore, we restrict ourselves to the first order predicates (*e.g.*, $\mathcal{P}_k :=$*BeforeEntity*["founded by"]). As a result, the objective function is simplified to: selecting a subset of $\mathcal{F}$ containing at most M snippet features and determine if each feature should be chosen as an indicative pattern in $\mathbf{P}$, such that $\mathcal{L}(\mathcal{F},\vec{w},\mathbf{P};\mathbf{y})$ is maximized.

With this simplification, we propose a greedy algorithm to learn PFNet. As PFNet shares a similar objective as general feature selection problem [71, 59], it is natural to adopt the forward selection idea to effectively select the feature set $\mathcal{F}$ and indicative pattern set $\mathbf{P}$. More specifically, starting with an empty feature set $\mathcal{F}$, we test every feature and greedily add the one which maximizes (5.12) into $\mathcal{F}$, which is similar to the existing feature selection works [71, 59]; whereas, with two key differences: first, the learner has to

Algorithm 2 Algorithm for Learning PFNet

Input: y_i, $f_k^{(e)}$, $\mathcal{F}_{\text{all}}$, M.
Output: $\mathcal{F} \subseteq \mathcal{F}_{\text{all}}$, $\mathbf{P}$, $\vec{w}$

$\mathcal{F}^{(0)} \leftarrow \emptyset$, $\mathbf{P}^{(0)} \leftarrow \emptyset$, $t \leftarrow 1$.
while $|\mathcal{F}| \leq M$ **do**
  $l_{\text{best}} \leftarrow -\infty, f_{\text{best}} \leftarrow \text{null}, \pi_{\text{best}} \leftarrow \text{null}$
  for $f_k^{(s)} \in \mathcal{F}_{\text{all}}$ **do**
    Optimize $\vec{w}$ to calculate $l^{(t)}(f_k, 0)$ according to Eq. 5.13
    if $l^{(t)}(f_k, 0) \geq l_{\text{best}}$ **then**
      $l_{\text{best}} \leftarrow l^{(t)}(f_k, 0)$, $f_{\text{best}} \leftarrow f_k$, $\pi_{\text{best}} \leftarrow 0$
    end if
    Optimize $\vec{w}$ calculate $l^{(t)}(f_k, 1)$ according to Eq. 5.14
    if $l^{(t)}(f_k, 1) \geq l_{\text{best}}$ **then**
      $l_{\text{best}} \leftarrow l^{(t)}(f_k, 1)$, $f_{\text{best}} \leftarrow f_k$, $\pi_{\text{best}} \leftarrow 1$
    end if
  end for
  $\mathcal{F}^{(t)} \leftarrow \mathcal{F}^{(t-1)} \cup \{f_{\text{best}}\}$
  if $\pi_{\text{best}} = 1$ **then**
    $\mathbf{P}^{(t)} \leftarrow \mathbf{P}^{(t-1)} \cup \{f_{\text{best}}\}$
  end if
  $t \leftarrow t + 1$
end while
return $\mathcal{F}$, $\mathbf{P}$, $\vec{w}$

search $\vec{w}$ to decide the best feature at each round; second, when a new feature is added, the learner will construct $\mathbf{P}$ immediately by deciding if the new feature would be chosen as an indicative pattern. The learning algorithm is summarized in Algorithm 2.

As Algorithm 2 shows, starting with an empty feature set $\mathcal{F}^{(0)}$ and an empty indicative pattern set $\mathbf{P}^{(0)}$, at the t-th round, the learner will search for the best $f^{(s)}$ and decide if it is an indicative pattern to maximize $\mathcal{L}(\mathcal{F}, \vec{w}, \mathbf{P}; \mathbf{y})$. Formally, such a searching process at the t-th round can be characterized by a sub-objective function $l^{(t)}(f^{(s)}, \pi)$, where $\pi = 1$ indicates $f^{(s)}$ is selected as an indicative pattern, and $\pi = 0$ as a common feature, defined as follows,

$$l^{(t)}(f^{(s)}, 0) := \max_{\vec{w}} \mathcal{L}(\mathcal{F}^{(t-1)} \cup \{f^{(s)}\}, \vec{w}, \mathbf{P}^{(t-1)}) \tag{5.13}$$

$$l^{(t)}(f^{(s)}, 1) := \max_{\vec{w}} \mathcal{L}(\mathcal{F}^{(t-1)} \cup \{f^{(s)}\}, \vec{w}, \mathbf{P}^{(t-1)} \cup \{f^{(s)}\}) \tag{5.14}$$

The best feature (denoted as f_{best}, together with π_{best}) which maximizes $l^{(t)}(f^{(s)}, \pi)$ will be added to $\mathcal{F}^{(t)}$, and $\mathbf{P}^{(t)}$ if π_{best} equals to 1. The iteration will stop until M features are collected.

To calculate Eq. 5.13 and Eq. 5.14 in each iteration, we use an inner loop to search the optimal $\vec{w}$ by

Dataset	Target Type	Query Num	Positive / Total Entity Num	Snippet Num
FounderOf	#person	371	473 / 20061	1033507
PublisherOf	#organization	323	329 / 19166	1488347
WriterOf	#person	669	993 / 46683	2111565
PlaceOfBirth	#location	350	350 / 24348	1376995
PlaceOfDeath	#location	350	350 / 23246	1105738
GraduateFrom	#organization	228	228 / 8559	97916

Figure 5.7: Dataset specifications.

gradient ascend, according to their gradients defined as follows,

$$\frac{\partial log\mathcal{L}}{\partial w_k^{(e)}} = \sum_{e_i \in \mathbf{e^{(P)}}} [y_i - P(y_i = 1|e_i, \mathbf{s_i})] f_k^{(e)}(e_i) \tag{5.15}$$

$$\frac{\partial log\mathcal{L}}{\partial w_k^{(s)}} = \sum_{e_i \in \mathbf{e^{(P)}}} [y_i - P(y_i = 1|e_i, \mathbf{s_i})] \sum_{s_{ij} \in \mathbf{s_i^{(P)}}} f_k^{(s)}(s_{ij}) \tag{5.16}$$

Note that such an optimization procedure is still inefficient since we have to search for the optimal $\vec{w}$ for every candidate feature. To further improve the efficiency, we decide to relax it: when testing a new feature $f_k^{(s)}$, we keep $\vec{w}$ from the last iteration unchanged, and only use the inner loop to find the optimal $w_k^{(s)}$ for the new feature. The whole vector $\vec{w}$ will be updated only after the best feature is added. Such a strategy avoid updating the whole vector $\vec{w}$ during feature testing.

5.5 Experiment

5.5.1 Experiment Setting

We chose around 100 million general English Web pages (about 20%) from the ClueWeb09 dataset as our testbed. To retrieve entities, we used our previously built system [82] (distributed over 30 nodes) as the entity-aware searcher to index the dataset. A list of entity types were extracted and indexed, including general entities (*e.g.*, #person, #organization and #location) extracted by Stanford NER toolkit [25].

When choosing the evaluation collection, we consider three criteria: first, the chosen collections should cover different target types to evaluate the model's capability over different types of relations; second, they should have different quality of evidence snippets in order to show the influcence of noise on ranking performance; third, they should also include less popular queries to validate the capability of PFNet on those collections with less redundancy.

Based on the above consideration, we collected six sets of different relations, two from Freebase (*FounderOf*()

and *PublisherOf*()), three from Wikipedia (*WriterOf*(), *PlaceOfBirth*() and *PlaceOfDeath*()), and one from the Mathematics Genealogy Project [1] (*GraduateFrom*()). As shown in Figure 5.7, they cover three different target types; in particular, the *GraduateFrom*() collection is much smaller than others, because the queries used in the collection are the names of graduate students, which occur much less frequently on the Web than the queries from Wikipedia and Freebase.

We can take the *FounderOf*() relation as an example to illustrate our procedure for collecting the evaluation data with entity-level annotations. Given the relation *FounderOf*("microsoft," {"bill gates," "paul allen,"}) we used the selected entry terms from this target relation as queries (*e.g.*, "microsoft") to search in our ClueWeb09 dataset, and retrieved all the entities matching the required target type (*e.g.*, #person for *FounderOf*()) together with the associated snippets. The retrieved entities will be labeled accordingly, *e.g.*, "bill gates" and "paul allen" are labeled as positive and all the others as negative.

PFNet has two free parameters, *i.e.*, λ and ε, to be tuned. We used 5-fold cross validation to find the optimal parameters through extensive experiments for each collection. We used $M = 10$ in all the datasets (later we would investigate the impact of feature size), and used 5-fold cross validation to confirm the confidence of the comparison with standard t-test.

5.5.2 Performance Comparison with Baselines

We employed five baseline methods to validate the performance of the proposed PFNet. First, to demonstrate the necessity of learning relation-specific ranking functions, we compared PFNet with EntityRank [18], a general ranking algorithm without distinguishing entity types. Second, as our task is formalized as a distantly supervised problem, we compared PFNet with Multi-Instance Learning (MIL) – a distant supervision model introduced in [63]. We designed two versions of MIL as baselines: MIL-All using all features, and MIL-IG using only top M snippet features chosen by the standard information gain criterion [77], to demonstrate the effectiveness of the feature selection component in PFNet. Third, since our task can also be viewed as a learning-to-rank problem, we compared PFNet with SVMRank, a state-of-the-art learning to rank model [35]. Similar to MIL, we employed both SVMRank-All and SVMRank-IG as baselines. The details of five baseline models are introduced as follows,

1. EntityRank [19]: this work aims at designing a general entity ranking function. Different from PFNet, EntityRank relies on user-specified keywords to represent the target relation (*e.g.*, "microsoft founder #person," where "founder" is treated as a keyword like "microsoft"), and thus, does not require any training data. To conquer the possible variants of keywords used to describe the relation, we employed

[1] http://genealogy.math.ndsu.nodak.edu/

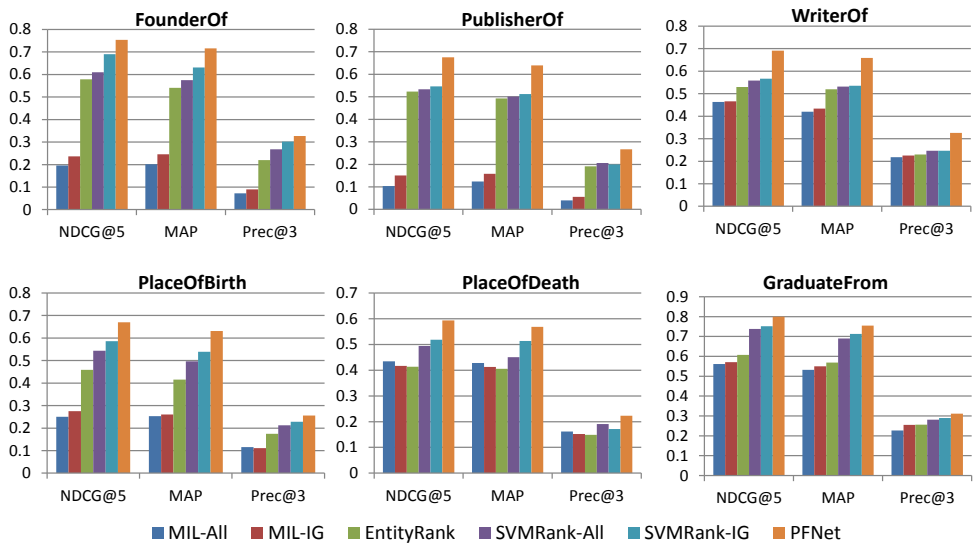


Figure 5.8: Ranking performance on different collections.

the patterns learned by PFNet as keywords instead of manually designating as in the original work.

2. Multi-instance Learning with All Features (MIL-All) [63]: this work shares similar motivation as ours, which aims to extract relations from text data according to distant supervision. To overcome noise, MIL assumes there has to be at least one positive snippet for a positive relation, and no positive snippet for a negative relation. Different from PFNet, MIL does not explicitly model redundancy. To make a fair comparison, we fed MIL with all features defined in PFNet.

3. Multi-instance Learning with Information Gain (MIL-IG): As our problem requires to use only a small number of features, MIL-IG uses a standard information gain criterion to choose the top M features. To calculate information gain for each feature, we assume all snippets in positive entities are positive and those in negative entities are negative. The chosen features are then fed into MIL for learning purpose.

4. SVMRank with All Features (SVMRank-All): this baseline treats the distantly supervised ranking problem as a standard learning to rank problem. To adapt SVMRank, we construct the feature vector for each entity by integrating all the associated snippet features into one feature vector. Such integration naturally leverages redundancy within the snippet contents. We used the implementation in [35] with all the default parameter settings.

5. SVMRank with Information Gain (SVMRank-IG): Similar to MIL-IG, this baseline only uses top M features chosen by information gain criterion.

Collection	*FounderOf*	*PublisherOf*	*WriterOf*
Evidence	62.7%	72%	76.7%
Collection	PlaceOfBirth	*PlaceOfDeath*	*GraduateFrom*
Evidence	76.7%	90.3%	49.4%

Figure 5.9: Noise percentage of snippets sampled from positive entities.

Since the relational entity search task is mostly search-oriented, we employ standard information retrieval performance metrics [32], e.g., NDCG@k, MAP and Precision@k, as our evaluation criteria. As Figure 5.7 shows, there are only one or two positive entities for most of queries, such as "*facebook founder*" and "*publisher of starcraft*," we calculate NDCG@5 and Precision@3 as the performance metrics. The ranking performance of our proposed PFNet and all the other five baseline methods on the six collections are shown in Figure 5.8.

The results demonstrate that PFNet outperforms all the baselines in six collections. In particular, PFNet achieves encouraging improvement of NDCG@5 and MAP against the runner-ups in the *PublisherOf* (NDCG@5 +23.7%, MAP +24.9%), *WriterOf* (NDCG@5 +22.0%, MAP +23.1%), *PlaceOfBirth* (NDCG@5 +14.4%, MAP +17.1%), *PlaceOfDeath* (NDCG@5 +14.4%, MAP +10.7%) collections (p-value$<$0.05 in all cases). We also observe that the improvement of PFNet decreases in the *GraduateFrom* (NDCG@5 + 6.3%, MAP + 5.9%) and *FounderOf* (NDCG@5 +9.1%, MAP +13.4%) collections. By analyzing the results, we find that PFNet's performance gain is closely related to the percentage of noisy snippets in the positive entities. To verify this, we sampled 300 snippets from positive entities in each collection, and manually labeled each snippet as evidence or noise, as demonstrated in Figure 5.9. The result shows that on those more noisy collections (more than 70% noisy snippets), *i.e.*, *PublisherOf*(), *WriterOf*(), *PlaceOfBirth*() and *PlaceOfDeath*(), PFNet achieves significant improvement; while for the collections of *FounderOf*() and *GraduateFrom*(), since they are less noisy, the baseline methods can already achieve satisfactory results. Such results confirm that noise filtering plays an important role in our distantly supervised ranking problem.

From the comparisons in Figure 5.8 we can clearly notice the advantage of PFNet over the baseline methods. MIL-All does not perform well in our data sets because it is developed for traditional information extraction task and heavily depends on a large number of high-precision low-recall features to distinguish noise. Without leveraging redundancy, it fails when only light-weighted features are available. Although EntityRank uses all the patterns learned by PFNet, it works in a relation-independent manner and does not work well. For the SVMRank-All baseline, though it models the concept of redundancy, it uses all snippets in training without distinguishing the noise. By comparing SVMRank-IG with SVMRank-All, and MIL-IG with MIL-All, we can find that the information-gain-based feature selection does improve the ranking performance; however, the improvement is limited. The reason is that when calculating the information

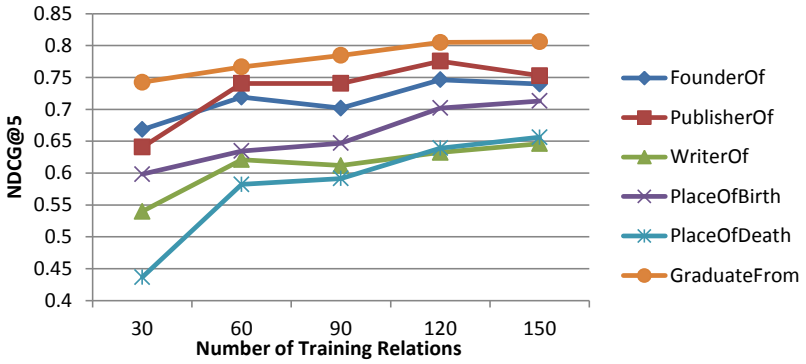


Figure 5.10: Ranking performance of PFNet with different size of training data.

gain of each feature, we assume all snippets in positive entities are positive, which makes information gain criterion vulnerable to noisy snippets. Comparing to SVMRank-IG and MIL-IG, PFNet explicitly depends on the automatically learned indicative patterns to filter out the noisy snippets and thus achieves better performance.

5.5.3 Influence of Redundancy on Ranking Performance

In this section, we will investigate the influence of redundancy on the model's ranking performance. In particular, we are interesting in two questions. First, in the training phase, how crucial the redundancy is for learning a robust and accurate model? Second, in the searching phase, will a ranking model's performance vary on the queries of different redundancy? Note that although five of our collections come from Wikipedia and Freebase, the collections still contain a lot of unpopular queries that are associated with only a small number of snippets.

To analyze how much data is needed in the training phase, we fixed 60 queries in each collection as testing data, and varied the number of training queries from 30 to 150. The ranking performance of PFNet over different size of training relations is demonstrated in Figure 5.10. The result shows that PFNet benefits from more training relations; whereas, the performance improvement differs in different collections. In particular, from 30 to 150 relations, PFNet achieves significant improvement in *PublisherOf* (NDCG@5 +17.4%), *WriterOf* (NDCG@5 +19.7%), *PlaceOfBirth* (NDCG@5 +19.2%) and *PlaceOfDeath* (NDCG@5 +50.3%) collections. While the improvement gets diminished in *FounderOf* (NDCG@5 +10.7%) and *GraduateFrom* (NDCG@5 +8.6%) collections. Referring to Figure 5.9, we notice that such difference is also highly related with the noise level of the collection. For collections containing less noisy snippets (*i.e.*, *FounderOf*() and *GraduateFrom*()), the learning task is relatively easier. Therefore, PFNet can achieve promising performance with a small number of training relations and does not benefit much from more training data; while for noisy collections, more training data is beneficial. In general, we observe that when the relation size increases to

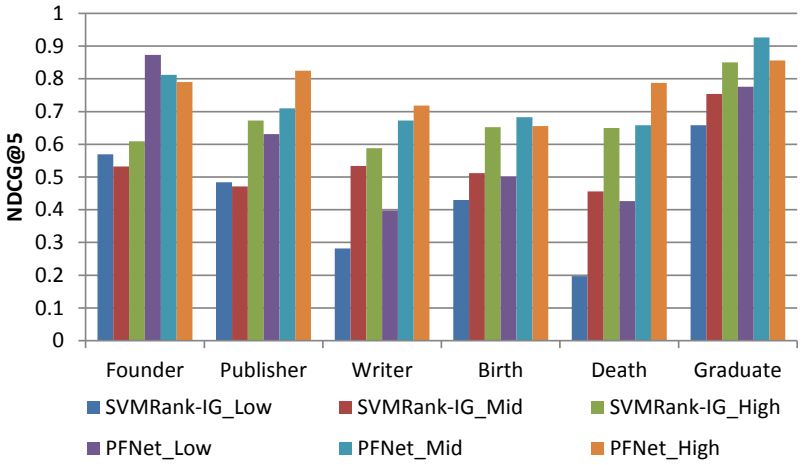


Figure 5.11: Ranking performance on queries of different popularity.

Data Set	*FounderOf*	*PublisherOf*	*WriterOf*
Covered Entities	21	15	13
Total Entities	21	20	22
Data Set	*PlaceOfBirth*	*PlaceOfDeath*	*GraduateFrom*
Covered Entities	16	13	20
Total Entities	20	20	20

Figure 5.12: The number of covered entities for low-popularity queries.

more than 120, the ranking performance becomes stable. It demonstrates that a median size of training data (*e.g.*, 120 queries) is generally sufficient for PFNet to estimate a good ranking model.

To evaluate the ranking performance of PFNet on queries of different redundancy, we classified queries into 3 categories: low-popularity (where positive entities have less than 10 snippets), medium-popularity (10 to 80 snippets) and high-popularity (more than 80 snippets). We sampled 20 queries from each category to construct 3 testing sets and used the remaining queries as training data. We used SVMRank-IG, which is the runner-up in Figure 5.8, as the baseline. The performance of PFNet and SVMRank-IG over queries of different popularity is demonstrated in Figure 5.11. The result fits our intuition: both PFNet and SVMRank-IG achieve better performance on high-popularity queries comparing to low-popularity ones. The explanation is intuitive: popular entities are more likely to contain evidence snippets compared with the uncommon entities. As a result, the redundancy ranking principle renders both PFNet and SVMRank-IG the capability of deriving more confident results for the popular entities.

However, exploiting redundancy does not necessarily lead to the conclusion that PFNet would completely fail for low-popularity queries. As Figure 5.11 shows, PFNet achieves comparable performance over low-popularity queries comparing to the high-popularity ones, *e.g.*, *GraduateFrom* (-8%) and *PlaceOfBirth* (-15%); while for the *FounderOf*() relation, low-popularity queries even performs better (+8%). By analyzing

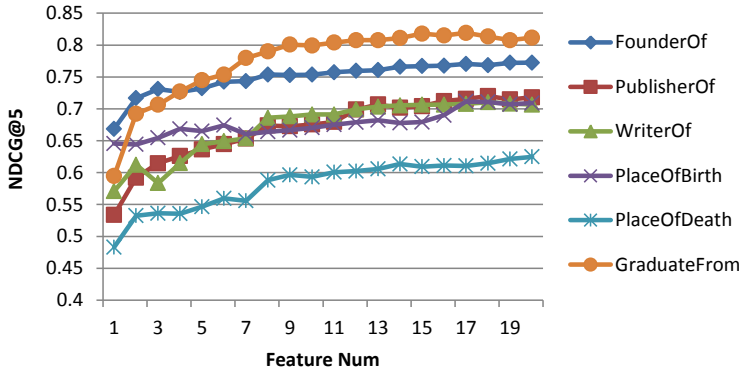


Figure 5.13: Ranking performance of PFNet with different feature number.

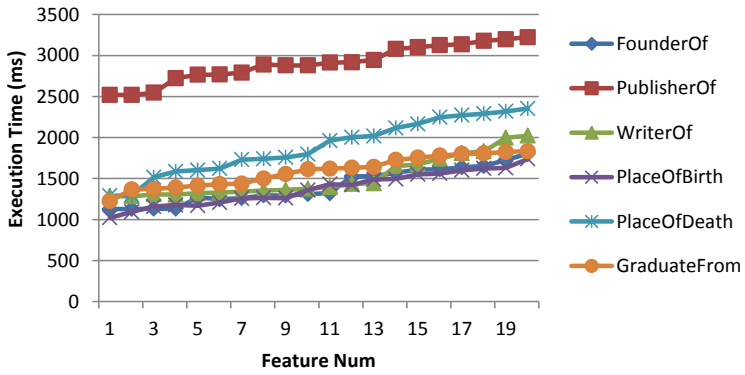


Figure 5.14: Average execution time per query with different feature number.

the result, we observe that PFNet achieves better performance on the collections with more covered entities (the entities which are associated with at least one evidence snippet matched by the identified indicative patterns). As demonstrated in Figure 5.12, both *FounderOf*() and *GraduateFrom*() have all low-popularity entities covered, where PFNet performs well over the low-popularity queries. And from another perspective, more importantly, such results demonstrate that the patterns identified by PFNet are general and effective, which are able to discover uncommon entities and achieve promising performance on the low-popularity queries as well.

By comparing PFNet with SVMRank-IG, we can find that PFNet outperforms SVMRank-IG in all categories and PFNet achieves more improvement over low-popularity and medium-popularity queries compared with high-popularity queries (average improvement on low-popularity queries: 45.9%, medium-popularity queries: 38.3%, high-popularity queries: 16.2%). The result demonstrates redundancy helps the model combat with noisy snippets, and thus even the baseline method (SVMRank-IG) could benefit from it. The comparison also demonstrates the capability of PFNet in handling low-popularity and medium-popularity queries.

Dataset	*FounderOf*	*PublisherOf*	*WriterOf*
Inverted Index	1310	2879	1374
Doc Checking	5692	6318	7604
Dataset	*PlaceOfBirth*	*PlaceOfDeath*	*GraduateFrom*
Inverted Index	1357	1796	1613
Doc Checking	4004	3939	2094

Figure 5.15: Efficiency comparison (in ms) between two feature fetching methods: inverted-index-based (10 patterns) and document-checking-based

5.5.4 Performance Comparison with Different Feature Size

In this section, we are going to investigate the effect of the number of features in our PFNet ranking framework. There are two important aspects to be assessed: first, how would number of chosen features affect the final ranking performance; second, how would the number of chosen features affect the execution efficiency.

Figure 5.13 shows PFNet's NDCG@5 performance with different number of features. We find that NDCG@5 improves a lot from 1 feature to 10 features: the improvement is over 3% for the *PlaceOfBirth*() relation, and over 10% for all other relations. While from 10 features to 20 features, the improvement gets diminished, less than 5% for all the relations. By analyzing the results, we find that the results are heavily affected by the "diversity" of a relation representation, *i.e.*, how many different patterns are used to characterize the relation. For example, the *PlaceOfBirth*() relation is usually described by a small number of patterns such as "born in" and "birth place," and therefore the model can achieve quite satisfactory results with less than 5 features. While for the *GrdauteFrom* relation, people usually do not explicitly mention "graduated from" but rather use a variety of other patterns such as "studied in," "got phd from" and so on. As the result, PFNet benefits more from using a larger set of features in this kind of relations.

In addition to the effect on the ranking performance, we also need to analyze the impact of the number of selected features on the ranking efficiency, because the more features we select the more indexes we have to check during the online execution phase. In this experiment, we sampled 60 queries from each data set and calculated the average execution time over all the collections in Figure 5.14. In PFNet, the overhead increases linearly with the size of selected features. By considering the performance improvement gained from the additional patterns (in Figure 5.13), a median size of indicative patterns (*e.g.*, $M = 10$) would be a good trade-off.

We also compared our choice of the inverted-index-base feature fetching approach to the document-checking-based approach, which was introduced in Section 5.2, in Figure 5.15. The result demonstrates that the document-checking approach is much less efficient than inverted index checking (about 2 to 3 times

Task	Query	1st Result	2nd Result	3rd Result
Founder Of	nike inc	philp h knight	bill bowerman	phil knight
	kompakt	wolfgang voigt	michael mayer	perlon
Place Of Birth	anne bancroft	anna maria	new york	bronx
	terrel davis	san diego	american	calfornia
Writer Of	the dock of the bay	michael bolton	steve cropper	otis redding
	game of death	bruce lee	robert clouse	john barry

Figure 5.16: Query example and top-3 returned entities (underlined entities are ground-truth from Freebase).

Task	Indicative Pattern
FounderOf	*Around*["founder"], *BeforeEntity*["founders"] *AfterQuery*["founded in"], *AfterEntity*["officer"]
PublisherOf	*BeforeEntity[published]*, *AfterEntity*["distribueur"] *BeforeEntity*["developed by"], *BeforeEntity*["released by"]
WriterOf	*Around*["written"], *BeforeEntity*["director"], *BeforeEntity*["writer"], *BeforeEntity*["composed by"]
PlaceOfBirth	*BeforeEntity*["born"], *AfterQuery*["grew"], *Around*["graduated"], *Around*["born on"]
PlaceOfDeath	*Around*["died"], *BeforeEntity*["of death"], *BeforeEntity*["home in"], *BeforeEntity*["died"]
GraduateFrom	*Around*["university"], *BeforeEntity*["phd thesis"], *BeforeEntity*["ph.d"], *BeforeEntity*["school of"]

Figure 5.17: Discovered indicative patterns.

slower). Such significant efficiency improvement confirms the necessity of choosing only indexable features in our online relational entity search task.

5.5.5 Case Study

To give an intuitive illustration of relational entity search task and diagnose the quantitative ranking performance, in Figure 5.16, we perform case studies by showing the top 3 entities returned by PFNet for some typical queries in our data set. We observe that PFNet can often return "positive" entities in front, but they are not listed in Freebase. For example, "bill bowerman," as a co-founder of "nike inc," is not recorded in Freebase, but PFNet ranks it at the 2nd position. The same case is for "michael bolton" to the query "the dock of the bay."

Besides, we also observe some limitations of PFNet. First, PFNet can not handle ambiguous queries: "wolfgang vogit" is ranked at the first place by PFNet for query "kompakt," however he was the founder of another "kompakt," a company different from the one founded by "michael mayer." Second, PFNet fails to distinguish the synonyms of the positive entities. For example, "philp h knight" is ranked higher than the ground-truth "phil knight" simply because the former is more frequently mentioned in the corpus; and with

respect to the query "terrel davis," "san diego," "california" and "american" are all correct answers but only differ in resolution. We can expect that some post-processing techniques, *e.g.*, entity name disambiguation, can further improve the ranking performance of PFNet.

Besides, since it is hard to directly evaluate the quality of the identified indicative patterns by PFNet, we list some of those patterns in Figure 5.17 for qualitative analysis purpose. From the result, we find that most of the patterns match our intuition of the tasks: for example, in the *PublisherOf*() relation, the model extracts "published by" and "released by" to capture the different types of "publisher" in Web pages. Some patterns that do not directly indicate the target relation are also discovered, *e.g.*, "grew," "graduated" for *PlaceOfBirth*() relation. And for the *WriterOf*() relation, PFNet also discovered patterns such as *BeforeEntity*["director"]. By looking into the data, we find that the data set contained many movie names as queries, and for a large portion of such movies, the director is also the writer of that movie (*e.g.*, "bruce lee" is the writer and director of the movie "game of death").

Chapter 6

Conclusion

In this chapter, I first summarize the contributions of my thesis. In order to extend my works towards developing a practical entity-centric search system, I further propose some research problems as my future direction.

6.1 Summary: Contributions of My Thesis

The richness of the Web has rendered itself as a huge database storing various types of entities. Observing the rapidly increasing entity-centric information needs from end users, I propose and study the interesting problem of entity-centric search, to facilitate different kinds of entity-related search operations.

According to whether the concept of entity appears in input query or output result, I propose to categorize entity search operations into three categories: 1) querying by entities, 2) querying for entities and 3) querying by and for entities. Specifically, in querying by entities, I study the entity-centric document filtering problem, which aims at finding relevant documents for entities that are characterized by their identification pages; in querying for entities, I propose to build a general data-oriented content query system to support "content querying" for finding various entity data in the text; in querying by and for entities, I study the relational entity search problem, which, given a query entity, tries to search other entities that match a desired relation.

This thesis aims at tackling these problems towards the goal of enabling entity-centric search. We summarize the challenges of these problems and our key insights which enable us to conquer the challenges.

First, for entity-centric document filtering, the key challenge lies in the fact that a keyword usually has very different importance for different entities, and the scorer has to appropriately transfer the keyword importance learned from training entities to unseen query entities. Based on the insight that keywords sharing some similar properties should have similar importance, we propose the idea of meta-features [80], and further extend it to the concept of feature decoupling [81], to realize bridging keywords across different entities. Our experiment results demonstrate the effectiveness of our proposed methods.

Second, for data-oriented content-query system, to support different types of entity-targeted application,

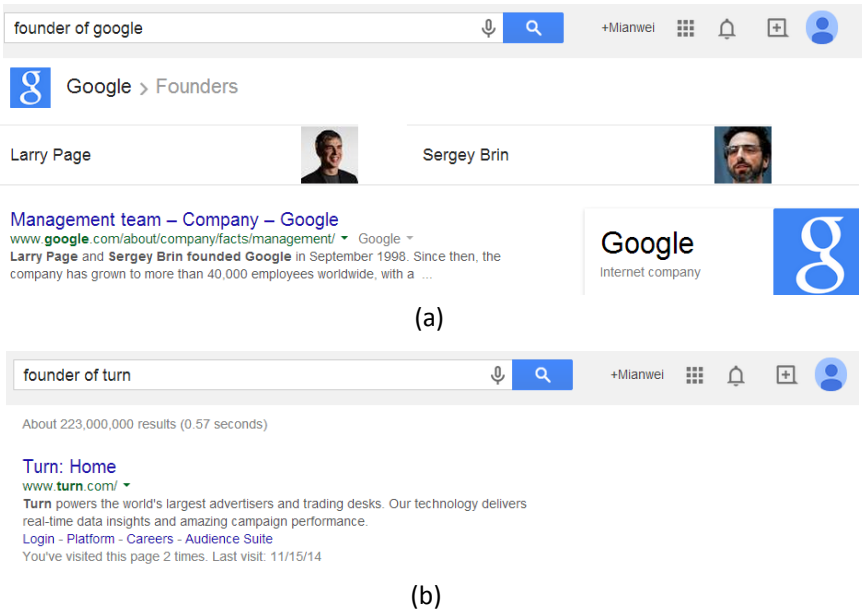


Figure 6.1: Google results for query: (a) "founder of google" (b) "founder of turn".

we are facing the challenge of designing a general framework which is capable of fulfilling their fundamental requirements: 1) extensible data types, 2) flexible contextual patterns and 3) customizable scoring function. To fulfill such requirements, I propose a relational-model based framework and a powerful content query language as our solution. The experiment result demonstrates that our query language is rather flexible and expressive, and our query processing is efficient with reasonable index overhead.

Third, for relational entity search, I exploit the redundancy of relation mentions for effective entity ranking. However, such redundancy is usually noisy, as we only have labels defined on the entity level, and obtaining detailed labels for each snippet manually is impractical. As our solution, we develop Pattern-based Filter Network (PFNet), a novel probabilistic graphical model, to balance the accuracy and efficiency requirements. The result validates the effectiveness of leveraging redundancy for entity ranking.

In summary, I study a set of entity-centric search problems in my thesis and propose novel techniques as the solutions. Through systematic evaluation, we demonstrate that entity-centric search is a promising direction towards improving user search experience.

6.2 Goal: A Practical Entity-Centric Search System

With our research works preparing solutions for tackling different types of entity-centric search operations, our ultimate goal is to build an entity-centric search system which could be integrated into modern search

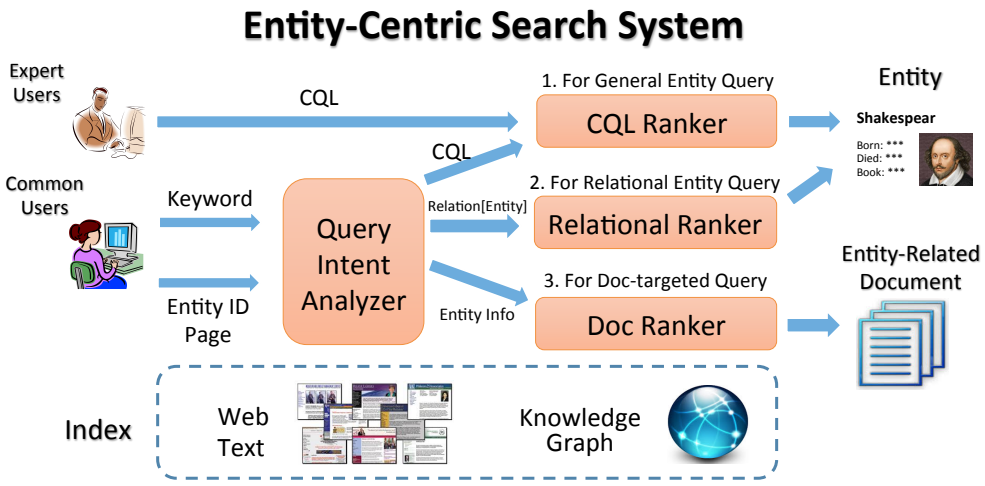


Figure 6.2: Entity-centric search framework.

engines (*e.g.*, Google, Bing) and easily used by end users.

Modern search engines nowadays still provide very limited functions to support entity-centric search. First, with the trend of accessing the internet using mobile devices, it becomes critically important for search engines to return desired entity information as query results, rather than require users to find answer by browsing many documents on a small screen. Although current search engines are developing towards such a direction, they can only return entities that are pre-indexed in knowledge bases as answers. For example, as shown in Figure 6.1, Google returns person names for "founder of google," but fails to answer "founder of turn," because Turn is a smaller company and does not have its Wikipedia page. Second, current search engines fail to support analytic queries, which require aggregating information across different Web pages to get the answer. For example, people might be interested in the most popular TV shows in the US or the most frequently discussed stock this week. In order to obtain the popularity information, a search engine needs to examine what TVs or stocks are mentioned in different Web sites, which could not be fulfilled by simple keyword search.

We believe that our research works could benefit current search engines in various aspects to resolve such limitations. As our ultimate goal, we plan to develop an entity-centric search framework based on our works, as shown in Figure 6.2.

In terms of input, the entity-centric search system would support various types of queries. Specifically, for expert users, our system would directly provide CQL as the open API, by which we can easily address the aforementioned analytic queries, *e.g.*, ranking TV show entities according to their occurrences in different Websites and stock entities according to how frequently they are mentioned in user reviews or comments.

For common users, our system would take keyword queries, which have been proved as the easiest way to represent user intents, as input. If users feel that using simple keywords is insufficient to characterize targeted entities, they can also adopt entities' identification pages, *e.g.*, Wikipedia pages, as queries.

In terms of ranking, the system has a set of rankers to tackle with different types of queries. For CQL queries inputted by expert users, they will be directly fed into the CQL ranker, which interprets and processes the CQL queries using the techniques introduced in Chapter 4. For other types of queries inputted by common users, the system has a query intent analyzer to analyze their intents. If the query is a general entity-targeted query, e.g., "the tallest building in the world", the analyzer will automatically translate it to a CQL query for entity retrieval; if the query searches for an entity that matches a certain relation with other entities (*e.g.*, "population of China"), the analyzer will translate the query to the form of *Relation*[*Entity*] (*e.g.*, *PopulationOf*[*China*]), and pass it to the relational ranker introduced in Chapter 5; if the query is to search documents about a particular entity, our system will apply our entity-centric document filtering technique discussed in Chapter 2 to retrieve relevant documents.

In terms of output, the rankers will retrieve data from the index storing Web pages and entity knowledge graphs, and return ranked entities or documents as results according to the types of queries. Specifically, for entity-targeted queries, in order for end users to better understand an entity answer, the search engine would, first, display text snippets to show how the entity results match the input query and, second, provide more information to describe the target entities.

Towards realizing such an entity-centric search framework, we need to further explore many research directions in our future work.

6.2.1 Future Work 1: Query Intent Analysis

In order to develop the system as shown in Figure 6.2, we need to build a query intent analyzer to explore the underlying entity-centric intents behind queries. For example, given a query "how heavy is Canon EOS Rebel T3i," the ranker needs to know that the target entity would be a number, while for query "top ten universities in computer science," it is asking for a university entity. Besides entity type, it is also important to analyze the entities and target relations mentioned in the query. In the aforementioned example, the ranker should identify that "Canon EOS Rebel T3i" and "computer science" are queries entities, while "how heavy" and "top ten" characterize the target relations.

The discovered entity-centric intent could be used to achieve more accurate document ranking. For the "how heavy is Canon EOS Rebel T3i" example, a relevant document is usually a descriptive page of "Canon EOS Rebel T3i," and thus the ranker should assign those documents that frequently mentions "Canon EOS

Rebel T3i" with higher score; while for keyword "how heavy" which represents the target relation, the scorer should translate it to appropriate keywords such as "weight," "pounds," and require them to appear around the target entity.

6.2.2 Future Work 2: Informative Entity Result

As Figure 6.2 displays, for entity-targeted queries, instead of returning a list of noun phrases, we will explore how to provide users with more informative entity results. Nowadays, many commercial search engines (e.g., Google, Bing) maintain a backend knowledge graph storing a large number of entities, and directly return knowledge base entities as search results when queries match entity names. We will study how to leverage knowledge graphs for more informative entity result representation; however, different from previous works, our information of entities is aggregated from Web pages, and therefore not limited to entities stored in knowledge graphs.

For popular entities, we can directly show their profile information from knowledge graphs. To achieve such a goal, the key challenge lies in entity disambiguation, *i.e.*, how to choose the correct entity when there exist multiple entities sharing the same name in knowledge graphs. For example, given an aggregated entity "apple," we need to decide whether it refers to "Apple (Company)" or "Apple (Fruit)" in knowledge graphs. The keywords used around the entity are usually very useful signals; *e.g.*, if entity "Apple" is surrounded by keyword "released," "iPod," ..., it should be linked to the company entity. We can also analyze the question itself to disambiguate entities; *e.g.*, given "fruit with modest vitamin C," we can infer that it refers to apple fruit by keyword "fruit."

While leveraging knowledge graph information to represent entity results is feasible when the targeted entities are popular, in many cases, our interested entities are not popular enough to have their profiles in knowledge graphs. In such cases, we will study how to automatically generate entity profile pages for representation; for example, given a researcher's name, we can list his/her research interest, graduated universities, advisors, *etc.*. Such a task is challenging, because: first, we need to identify that the person name belongs to the researcher category, and recognize that "research interest" and "graduated university" will be useful attributes in the research category; second, for each attribute, we need to further extract their answers from searching results, *e.g.*, "UIUC" for "graduated university."

6.2.3 Future Work 3: Other Types of Entity-Centric Search

Although Figure 6.2 only shows three types of entity-centric search, in the future work, we will explore more other types of search operations.

First, in terms of input, when a query involves one entity, people are, as we have discussed, interested in the entity's information, whereas, when two or more entities are involved, we usually search their existing relations. For example, given a query "iPhone 6 and Samsung Galaxy S5," the search engine should be able to summarize their commonality and respective pros or cons in the result; for query "Martin Freeman and Amanda Abbington," the result should point out that they are spouses and co-star in some movies. For such relation-targeted queries, we will study how to discover their relations from their co-occurring snippets, and summarize them for better representation.

Second, in terms of output, in some cases, people are not satisfied with a set of entities but want to see how they are connected with other entities, *i.e.*, results represented by a network. For example, when academic researchers are surveying topics, they usually are not just interested in one or two papers, but want to reach other related topics, authors and their publications; it would be useful to represent them a network consisting of topics, authors and papers. As another example, when people want to buy a product, they would like to compare their specification with other related products and their prices in different stores; therefore, the ideal result should be a network connecting different products, stores and representing our required information.

In summary, this thesis studies a set of entity-centric search operations, proposes novel techniques and systematically evaluates our models on large datasets. In our future work, I will continue to explore more research problems on entity-centric search, towards improving the result quality of modern search engines and the search experience of end users.

References

[1] Trec knowledge base acceleration 2012, http://trec-kba.org/kba-ccr-2012.shtml.

[2] S. Abney, M. Collins, and A. Singhal. Answer extraction. In *Proceedings of the 6th Conference on Applied Natural Language Processing (ANLC)*, pages 296–301. Association for Computational Linguistics, 2000.

[3] Sanjay Agrawal, Kaushik Chakrabarti, Surajit Chaudhuri, and Venkatesh Ganti. Scalable ad-hoc entity extraction from text collections. *Proceedings of the VLDB Endowment*, 1(1), 2008.

[4] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. *Advances in Neural Information Processing Systems (NIPS)*, pages 577–584, 2003.

[5] S. Banerjee, S. Chakrabarti, and G. Ramakrishnan. Learning to rank for quantity consensus queries. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 243–250. ACM, 2009.

[6] Michael Bendersky and W Bruce Croft. Discovering key concepts in verbose queries. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 491. ACM, 2008.

[7] Michael Bendersky, Donald Metzler, and W Bruce Croft. Learning concept importance using a weighted dependence model. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM)*, pages 31–40. ACM, 2010.

[8] Michael Bendersky, Donald Metzler, and W Bruce Croft. Parameterized concept weighting in verbose queries. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 605–614. ACM, 2011.

[9] D.M. Blei and J.D. McAuliffe. Supervised topic models. *arXiv preprint arXiv:1003.0783*, 2010.

[10] J. Blitzer, M. Dredze, and F. Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 45, page 440, 2007.

[11] E. Brill, S. Dumais, and M. Banko. An analysis of the askmsr question-answering system. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 10, pages 257–264. Association for Computational Linguistics, 2002.

[12] Michael J. Cafarella. Extracting and querying a comprehensive web database. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2009.

[13] Michael J. Cafarella, Christopher Re, Dan Suciu, and Oren Etzioni. Structured querying of web text data: A technical challenge. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR)*, 2007.

[14] M.J. Cafarella and O. Etzioni. A search engine for natural language applications. In *Proceedings of the 14th International Conference on World Wide Web (WWW)*, pages 442–452. ACM New York, NY, USA, 2005.

[15] Y. Cao, J. Xu, T.Y. Liu, H. Li, Y. Huang, and H.W. Hon. Adapting ranking SVM to document retrieval. In *Proceedings of the 29th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 186–193. ACM, 2006.

[16] Claudio Carpineto and Giovanni Romano. A survey of automatic query expansion in information retrieval. *ACM Computing Surveys (CSUR)*, 44(1):1, 2012.

[17] Soumen Chakrabarti, Kriti Puniyani, and Sujatha Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *Proceedings of the 15th International Conference on World Wide Web (WWW)*, pages 717–726, 2006.

[18] T. Cheng and K.C.C. Chang. Entity Search Engine: Towards Agile Best-Effort Information Integration over the Web. *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 108–113, 2007.

[19] T. Cheng, X. Yan, and K.C.C. Chang. EntityRank: searching entities directly and holistically. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, pages 387–398. VLDB Endowment, 2007.

[20] C.L.A. Clarke, G.V. Cormack, and T.R. Lynam. Exploiting redundancy in question answering. In *Proceedings of the 24th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 358–365. ACM, 2001.

[21] W. Dai, G.R. Xue, Q. Yang, and Y. Yu. Co-clustering based classification for out-of-domain documents. In *Proceedings of the 13th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, volume 12, pages 210–219, 2007.

[22] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.M. Popescu, T. Shaked, S. Soderland, D.S. Weld, and A. Yates. Web-scale information extraction in knowitall:(preliminary results). In *Proceedings of the 13th International Conference on World Wide Web (WWW)*, pages 100–110. ACM, 2004.

[23] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall. In *Proceedings of the 13th International Conference on World Wide Web (WWW)*, 2004.

[24] A.A.T. Evgeniou and M. Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems (NIPS)*, volume 19, page 41. MIT Press, 2006.

[25] J.R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 363–370. Association for Computational Linguistics, 2005.

[26] J.R. Frank, M. Kleiman-Weiner, D.A. Roberts, F. Niu, C. Zhang, and Re C. Building an entity-centric stream filtering test collection for trec 2012. In *Proceedings of the 21st Text Retrieval Conference (TREC)*, 2012.

[27] J.H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[28] Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–274. ACM, 2009.

[29] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[30] R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer, and D.S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 541–550. Association for Computational Linguistics, 2011.

[31] A. Huang. Similarity measures for text document clustering. In *Proceedings of the 6th New Zealand Computer Science Research Student Conference (NZCSRSC)*, pages 49–56, 2008.

[32] Kalervo Jarvelin and Jaana Keklinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, October 2002.

[33] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, pages 137–142, 1998.

[34] T. Joachims. Making large scale svm learning practical. 1999.

[35] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 217–226. ACM, 2006.

[36] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the 25th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 133–142. ACM, 2002.

[37] Ghassan Kanaan, Riyad A Shalab, Sameh Ghwanmeh, and Basel B Ismail. Interactive and automatic query expansion: A comparative study with an application on arabic. *American Journal of Applied Sciences*, 5(11):1433, 2008.

[38] Gjergji Kasneci, Fabian M. Suchanek, Georgiana Ifrim, Maya Ramanath, and Gerhard Weikum. Naga: Searching and ranking knowledge. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*, 2008.

[39] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[40] Ravi Kumar and Andrew Tomkins. A characterization of online browsing behavior. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 561–570. ACM, 2010.

[41] Giridhar Kumaran and Vitor R Carvalho. Reducing long queries using query quality predictors. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 564–571. ACM, 2009.

[42] Oren Kurland, Lillian Lee, and Carmel Domshlak. Better than the real thing?: iterative pseudo-query processing using cluster-based language models. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26. ACM, 2005.

[43] C. Kwok, O. Etzioni, and D.S. Weld. Scaling question answering to the web. *ACM Transactions on Information Systems (TOIS)*, 19(3):242–262, 2001.

[44] Matthew Lease. An improved markov random field model for supporting verbose queries. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 476–483. ACM, 2009.

[45] Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in Neural Information Processing Systems (NIPS)*, pages 897–904, 2007.

[46] Tao Li, Vikas Sindhwani, Chris Ding, and Yi Zhang. Knowledge transformation for cross-domain sentiment classification. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 716–717. ACM, 2009.

[47] Jimmy J. Lin and Boris Katz. Question answering from the web using knowledge annotation and knowledge mining techniques. In *Proceedings of the 12th ACM Conference on Information and Knowledge Management (CIKM)*, 2003.

[48] Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proc. of SIGIR Workshop*, pages 3–10, 2007.

[49] X. Liu and H. Fang. Entity profile based approach in automatic knowledge finding. In *Proceeding of the 21st Text Retrieval Conference (TREC)*, 2012.

[50] Yuanhua Lv and ChengXiang Zhai. Adaptive relevance feedback in information retrieval. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)*, pages 255–264. ACM, 2009.

[51] B. Magnini, M. Negri, R. Prevete, and H. Tanev. Is it the right answer?: exploiting web redundancy for answer validation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 425–432, 2002.

[52] Paşca Marius, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Organizing and searching the world wide web of facts - step one: the one-million fact extraction challenge. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 2006.

[53] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1003–1011. Association for Computational Linguistics, 2009.

[54] N Okazaki. Liblbfgs: a library of limited-memory broyden-fletcher-goldfarb-shanno (l-bfgs). *URL http://www. chokkan. org/software/liblbfgs/index. html*, 2011.

[55] S.J. Pan, J.T. Kwok, and Q. Yang. Transfer learning via dimensionality reduction. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI)*, volume 2, pages 677–682, 2008.

[56] S.J. Pan, X. Ni, J.T. Sun, Q. Yang, and Z. Chen. Cross-domain sentiment classification via spectral feature alignment. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 751–760. ACM, 2010.

[57] S.J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(10):1345–1359, 2010.

[58] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[59] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1226–1238, 2005.

[60] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010.

[61] Ganesh Ramakrishnan, Sreeram Balakrishnan, and Sachindra Joshi. Entity annotation using inverse index operations. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2006.

[62] Frederick Reiss, Sriram Raghavan, Rajasekar Krishnamurthy, Huaiyu Zhu, and Shivakumar Vaithyanathan. An algebraic approach to rule-based information extraction. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*, 2008.

[63] S. Riedel, L. Yao, and A. McCallum. Modeling relations and their mentions without labeled text. *Machine Learning and Knowledge Discovery in Databases*, pages 148–163, 2010.

[64] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *NIST Special Publication SP*, pages 109–109, 1995.

[65] Joseph John Rocchio. Relevance feedback in information retrieval. 1971.

[66] Ian Ruthven. Re-examining the potential effectiveness of interactive query expansion. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in Informaion Retrieval*, pages 213–220. ACM, 2003.

[67] Ruslan Salakhutdinov and Geoffrey E Hinton. Replicated softmax: an undirected topic model. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1607–1614, 2009.

[68] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. *Readings in information retrieval*, 24(5), 1997.

[69] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web (WWW)*, pages 285–295. ACM, 2001.

[70] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory.

[71] S.D. Stearns. On selecting features for pattern classifiers. In *Proceedings of the 3rd International Joint Conference on Pattern Recognition*, pages 71–75, 1976.

[72] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[73] Vladimir Vapnik. Statistical learning theory, 1998.

[74] L. Weng, Z. Li, R. Cai, Y. Zhang, Y. Zhou, L.T. Yang, and L. Zhang. Query by document via a decomposition-based two-level retrieval approach. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2011.

[75] Minji Wu and Amelie Marian. Corroborating answers from multiple web sources. In *Proceedings of the 10th International Workshop on the Web and Databases (WebDB)*, 2007.

[76] Y. Yang, N. Bansal, W. Dakka, P. Ipeirotis, N. Koudas, and D. Papadias. Query by document. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining (WSDM)*, pages 34–43. ACM, 2009.

[77] Y. Yang and J.O. Pedersen. A comparative study on feature selection in text categorization. In *Machine Learning Internetional Workshop Then Conference*, pages 412–420. Morgan Kaufmann Publishers, Inc., 1997.

[78] B. Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, page 114. ACM, 2004.

[79] Wenyi Zhao, Arvindh Krishnaswamy, Rama Chellappa, Daniel L Swets, and John Weng. Discriminant analysis of principal components for face recognition. In *Face Recognition*, pages 73–85. Springer, 1998.

[80] Mianwei Zhou and Kevin C. Chang. Entity-centric document filtering: Feature mapping through meta-features. In *Proceedings of the 22nd ACM Conference on Information and Knowledge Management (CIKM)*. ACM, 2013.

[81] Mianwei Zhou and Kevin C. Chang. Unifying learning to rank and domain adaptation: Enabling cross-task document scoring. In *Proceedings of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2014.

[82] Mianwei Zhou, Tao Cheng, and Kevin C. Chang. Data-oriented content query system: searching for data into text on the web. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM)*, pages 121–130. ACM, 2010.

[83] Mianwei Zhou, Hongning Wang, and Kevin C. Chang. Learning to rank from distant supervision: Exploiting noisy redundancy for relational entity search. In *Proceedings of the 29th International Conference on Data Engineering (ICDE)*, pages 829–840. IEEE, 2013.

[84] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer, 2008.